ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ

ΒΑΘΙΑ ΜΑΘΗΣΗ

ΟΝΟΜΑΤΕΠΩΝΥΜΟ: Πορλού Χάιδω

ΣΧΟΛΗ: ΗΜΜΥ ΑΠΘ

ΑΕΜ: 9372

ΕΡΓΑΣΙΑ 1

MULTI LAYER PERCEPTRON (MLP)

Εισαγωγή

Η 1η εργασία που μου ανατέθηκε στο μάθημα Νευρωνικά Δίκτυα – Βαθιά Μάθηση αφορά ένα απλό πρόβλημα ταξινόμησης (classification) με τη χρήση ενός Multi Layer Perceptron (MLP), που εκπαιδεύεται με τον αλγόριθμο back propagation (ενημέρωση των συναπτικών βαρών μετά από την αξιολόγηση του μοντέλου, τυπικά σε κάθε επανάληψη εκπαίδευσης).

Όσον αφορά την εξαγωγή χαρακτηριστικών, επιλέχθηκε η χρήση όλης της εισόδου (ολόκληρης της 32x32 εικόνας), μιας και δεν υπήρχαν προβλήματα υπολογιστικής ισχύος και είναι μία λογική επιλογή για τη δημιουργία ενός απλού MLP για τους σκοπούς της εργασίας.

Σαν dataset επιλέχθηκε το cifar-10 dataset, το οποίο περιλαμβάνει 60,000 32x32 εικόνες που μπορούν να κατηγοριοποιηθούν σε 10 κλάσεις: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. Ο δικός μας στόχος είναι να μπορούμε να προβλέψουμε την κλάση/όνομα στην οποία κάθε εικόνα αντιστοιχεί, εξού η ονομασία του προβλήματος (ταξινόμηση). Σχολιάζεται πως το dataset είναι ήδη χωρισμένο σε training και testing υποσύνολα, με 50,000 και 10,000 στοιχεία αντίστοιχα.

Η επιλογή του συγκεκριμένου dataset έγινε διότι είναι εισαγωγικό όσον αφορά τον τομέα της ταξινόμησης εικόνων και σχετικά εύκολα ταξινομήσιμο και «με το μάτι», το οποίο βοήθησε αναμφισβήτητα στην την καλύτερη κατανόηση της διαδικασίας εκπαίδευσης, λόγω της «οπτικοποίησης» του όλου προβλήματος.

Πρώτη επαφή με το πρόβλημα και κατηγοριοποιητές k-nearest neighbor & nearest centroid

Η πρώτη επαφή με το dataset cifar-10 έγινε με την χρήση έτοιμων μοντέλων για την κατηγοριοποίηση k-nearest neighbor και nearest centroid. Σημειώνεται εδώ πως η εργασία υλοποιήθηκε σε python, και πιο συγκεκριμένα με την deep learning αρχιτεκτονική και τις βιβλιοθήκες keras.

Ο πρώτος κατηγοριοποιητής που χρησιμοποιήθηκε είναι ο knn (k-nearest neighbor), η λειτουργία του οποίου αναλύεται παρακάτω. Πρώτα, γίνεται η επιλογή της σταθεράς k. Οι τιμές που χρησιμοποιήθηκαν στην υλοποίηση της εργασίας είναι οι k=1 και k=3.

Στη συνέχεια, για την εκπαίδευση του μοντέλου χρησιμοποιούνται τα 50,000 στοιχεία που αντιστοιχούν στο training set (εικόνες 32x32x3 επειδή είναι έγχρωμες εικόνες), αφού γίνει μορφοποίηση αυτών σε vectors 3072 στοιχείων. Η μορφοποίηση αυτή συμβαίνει ώστε να μπορεί να υπολογιστεί η ευκλείδεια απόσταση κάθε στοιχείου με το/τα κοντινότερό/ά του. Ο knn λοιπόν κατηγοριοποιεί τις εικόνες του testing set υπολογίζοντας (μέσω της ευκλείδειας απόστασης) τους k κοντινότερους γείτονες της, και επιλέγει την κλάση της με βάση την πλειοψηφία στους γείτονες αυτούς.

Στη δική μας υλοποίηση, όπως προαναφέρθηκε, επιλέχθηκε k=1 γείτονας και k=3 γείτονες. Τα αποτελέσματα δεν ήταν ιδιαίτερα θετικά, κάτι αναμενόμενο, καθώς ο knn δεν είναι γνωστή καλή επιλογή στην κατηγοριοποίηση εικόνων. Αυτό συμβαίνει διότι η μετατροπή μιας εικόνας σε γραμμικό vector αφαιρεί ένα αρκετά σημαντικό κομμάτι πληροφορίας, όπως τη συσχέτιση μεταξύ γειτονικών πίξελ, τις απότομες αλλαγές χρώματος και φωτεινότητας κτλ.

Ο δεύτερος κατηγοριοποιητής που χρησιμοποιήθηκε είναι ο nearest centroid. Η διαχείριση των δεδομένων όσον αφορά αυτή τη διαδικασία είναι η ίδια (δημιουργία vector), ο τρόπος επιλογής της κλάσης όμως διαφέρει ελαφρώς. Πιο συγκεκριμένα, κατά τη διαδικασία της εκπαίδευσης, υπολογίζεται το average «κέντρο» της κάθε κλάσης (εξού και centroid), κι έτσι η κάθε κλάση έχει τη δική της προσδιορισμένη τιμή/centroid. Έτσι, όταν μία νέα εικόνα πρέπει να κατηγοριοποιηθεί, η τιμή της συγκρίνεται με όλα τα διαθέσιμα κέντρα των κλάσεων, και επιλέγεται η κλάση με τη μικρότερη ευκλείδεια απόσταση από το νέο σημείο.

Παρόμοια προβλήματα με τον knn παρουσιάζονται και στον αλγόριθμο nearest neighbor, οπότε τα αποτελέσματα του ήταν εξίσου μέτρια. Παρ'αυτά, τα αποτελέσματα όσον αφορά το accuracy (απόδοση) και τον χρόνο των 2 κατηγοριοποιητών παρουσιάζονται παρακάτω:

	time	**accuracy**
k-nearest neighbor (k=1)	21.1209619 seconds	0.3539
k-nearest neighbor (k=3)	22.2508304 seconds	0.3303
nearest centroid	1.37121844 seconds	0.2774

Σχολιάζεται πως το accuracy δεν είναι αρκετά καλό σε κανένα από τα μοντέλα μας, είναι όμως αισθητά καλύτερο στον κατηγοριοποιητή nearest neighbor. Από τα δύο k που επιλέξαμε, καλύτερη απόδοση έχουμε με k=1, κάτι το οποίο δεν είναι μη αναμενόμενο, καθώς καμιά φορά η προσθήκη παραπάνω γειτόνων μπορεί να οδηγήσει την κατηγοριοποίηση σε λανθασμένα μονοπάτια, λόγω του μεγάλου εύρους δεδομένων. Ειδικά σε τόσο μικρές αποδόσεις, στην ουσία η επιλογή πολλών γειτόνων έναντι λιγότερων αυξάνει την πιθανότητα ύπαρξης wrongly classified γειτόνων, η οποία σαν «ντόμινο» οδηγεί σε περαιτέρω λανθασμένη κατηγοριοποίηση. Τέλος, ο χρόνος εκτέλεσης του nearest centroid κατηγοριοποιητή είναι πολύ μικρότερος από τον knn, οπότε αν σε κάποια ιδιαίτερη εφαρμογή μας ενδιέφερε περισσότερο ο χρόνος σε σχέση με την απόδοση (ή αντίστοιχα σε κάποιο dataset με καλύτερη απόδοση), θα μπορούσε να ληφθεί υπόψιν.

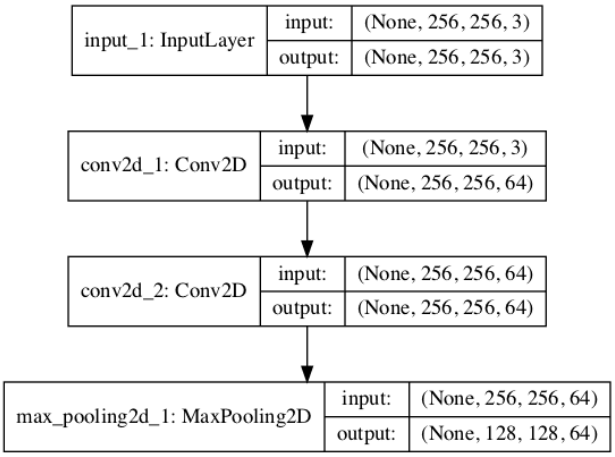
Διερεύνηση απόδοσης μοντέλου με διαφοροποιήσεις στο σχεδιασμό και τη διαδικασία εκπαίδευσης

Σε αυτό το κομμάτι της εργασίας καλούμαστε να μελετήσουμε τη λειτουργία ενός Multi Layer Perceptron, κάνοντας αλλαγές, ενδεικτικά, στον αριθμό νευρώνων που ανήκουν σε κάθε

στρώμα, στις ρυθμίσεις του optimizer της εκπαίδευσης, στο batch size, καθώς και στις διάφορες μεθόδους αντιμετώπισης της υπερεκπαίδευσης.

Η διαδικασία έχει ως εξής:

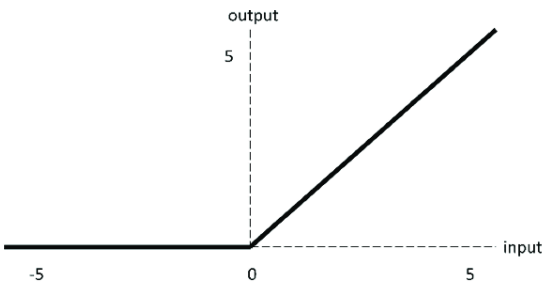
Αρχικά, το πρώτο βήμα που καλούμαστε να ολοκληρώσουμε είναι η δημιουργία του μοντέλου πάνω στο οποίο θα εκπαιδευτεί το dataset που επιλέξαμε. Το μοντέλο αυτό επιλέχθηκε να περιλαμβάνει 2 (dense) layers (σαν πρώτη ιδέα). Επειδή όμως χρησιμοποιούμε ένα dataset για κατηγοριοποίηση εικόνων, η οποία είναι γνωστό πως παρουσιάζει σχετικά κακά αποτελέσματα σε απλά MLP (βλ. σχήματα παρακάτω), αποφασίστηκε να προστεθούν 3 ακόμα layers (ένα μικρό convolutional neural network) πριν από το MLP, τα οποία αποτελούν ένα VGG block, όπως στην παρακάτω εικόνα:



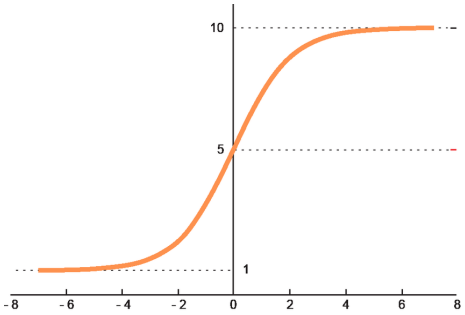
Στην ουσία, αναφερόμαστε σε 2 συνελικτικά layers που δημιουργούνται με συνέλιξη της εισόδου με 3x3 φίλτρα (με zero padding για να κρατήσουμε το μέγεθος της εισόδου), και στη συνέχεια σε ένα max pooling layer, σκοπός του οποίου είναι να μειωθεί στο μισό το «resolution» της εικόνας. Περιληπτικά, ο λόγος χρήσης της συνέλιξης είναι η καλύτερη διακριτική ικανότητα της (καθώς το φίλτρο «σύρεται» πάνω σε όλη την εικόνα, και «μαζεύει» περισσότερη πληροφορία, η οποία εμφανίζεται στην έξοδο). Σημειώνεται πως αυτό το κομμάτι δεν αποτελεί τη βάση της εργασία, οπότε προσπάθησα να το ενσωματώσω και να το καταλάβω όσο μπόρεσα με την αντίστοιχη βιβλιογραφία, αλλά όχι σαν «αυτοσκοπό». Ο λόγος που ενσωματώθηκε είναι επειδή αποτελεί οριακά αυτονόητη πρακτική για image classification, ώστε να μελετηθεί στη συνέχεια σωστά η απόδοση του MLP.

Αφού εισάγουμε το VGG, συνεχίζουμε στην σχεδίαση των layers που θα ακολουθήσουν. Αρχικά, προσθέτουμε ένα Flatten layers, ώστε να είναι πλέον οι εικόνες μας vectors ενός συγκεκριμένου μεγέθους, και στη συνέχεια προσθέτουμε τα 2 dense layers που αναφέρθηκαν στην αρχή. Η συνάρτηση ενεργοποίησης του επιλέχθηκε να είναι η ReLU (κλασσική επιλογή), και ο kernel initializer (για την αρχικοποίηση των βαρών στους νευρώνες) ο he uniform.

ReLU



softmax



Τέλος, η συνάρτηση ενεργοποίησης της εξόδου (το layer της οποία έχει φυσικά 10 νευρώνες, καθώς τόσες είναι οι κλάσεις στις οποίες κατηγοριοποιούμε τις εικόνες) είναι η softmax.

Μετά το πέρας της σχεδίασης και προσδιορισμού του μοντέλου, προχωράμε στην είσοδο/φόρτωση των δεδομένων, εδώ με τη βοήθεια της tensorflow. Τα δεδομένα μας (από το cifar 10) είναι ήδη, όπως προαναφέρθηκε, ήδη χωρισμένα σε training και testing sets, των 50,000 και 10,000 στοιχείων αντίστοιχα.

Αφού τα κανονικοποιήσουμε, το επόμενο βήμα είναι ο ορισμός του optimizer για την διαδικασία της εκπαίδευσης, εδώ ο SGD με διαφορετικά learning rates που θα παρουσιαστούν παρακάτω. Η επιλογή του SGD έγινε λόγω της σταθερότητας που προσφέρει, καθώς και επειδή μπορεί να θεωρηθεί ως ο πιο «κλασσικός» optimizer σε τέτοιου είδους προβλήματα.

Η αντικειμενική συνάρτηση προς βελτιστοποίηση που επιλέχθηκε είναι η categorical cross-entropy, η οποία χρησιμοποιείται κατά κόρον σε προβλήματα κατηγοριοποίησης, καθώς παράγεται μία κατανομή πιθανοτήτων για τις κλάσεις στο στρώμα εξόδου. Στην ουσία δηλαδή, για κάθε πιθανή κλάση υπάρχει μια πιθανότητα η είσοδος να ανήκει σε αυτή, και η μεγαλύτερη πιθανότητα οδηγεί στην κατηγοριοποίηση της στην αντίστοιχη κλάση. Αναφέρεται επίσης πως η μετρική αξιολόγησης εδώ είναι το accuracy (απόδοση), αλλά στη συνέχεια θα αναφερθούν και άλλες μετρικές με τη χρήση της βιβλιοθήκης sklearn.

Αφού ορισθεί το μοντέλο και όλοι οι παράγοντες που θα βοηθήσουν στην εκπαίδευση, προχωράμε στην εκπαίδευση αυτή καθ'αυτή (model.fit). Επιλέχθηκε για την πλειοψηφία των επαναλήψεων να δουλέψουμε με 50 epochs (επαναλήψεις/περάσματα των δεδομένων) και batch size = 64, ώστε να έχουμε ένα καλό trade off ταχύτητας και ακρίβειας, και να αποφευχθεί η υπερεκπαίδευση.

Τέλος, μετά την εκπαίδευση του μοντέλου προχωράμε στο evaluation αυτού, μέσω του οποίου βλέπουμε την απόδοση του στο testing set που δεν έχει ξαναδεχθεί σαν είσοδο. Στη συνέχεια, αφού ορίσουμε τα ονόματα όλων των κλάσεων της εξόδου σε έναν πίνακα με strings, προχωράμε στην ενδεικτική πρόβλεψη 10 εικόνων από το testing set, παραδείγματα της οποίας (σωστά και λανθασμένα) θα παρουσιαστούν παρακάτω. Παρακάτω θα παρουσιαστούν επίσης διαγράμματα για την απόδοση (accuracy) και το σφάλμα (loss) του κάθε διαφορετικού μοντέλου (κατά τη διάρκεια περασμάτων/epochs), καθώς και το classification report από τη βιβλιοθήκη sklearn, που παρουσιάζει τις μετρικές accuracy, precision, recall και f-measure για κάθε κλάση ξεχωριστά.

Στον πίνακα που ακολουθεί φαίνονται αναλυτικά τα διαφορετικά πειράματα που έγιναν με διαφοροποιήσεις στα εξής:

- αριθμός νευρώνων στα hidden layers
- batch size

- learning rate του optimizer

καθώς και μέθοδοι αποφυγής της υπερεκπαίδευσης (περισσότερη ανάλυση στη συνέχεια):

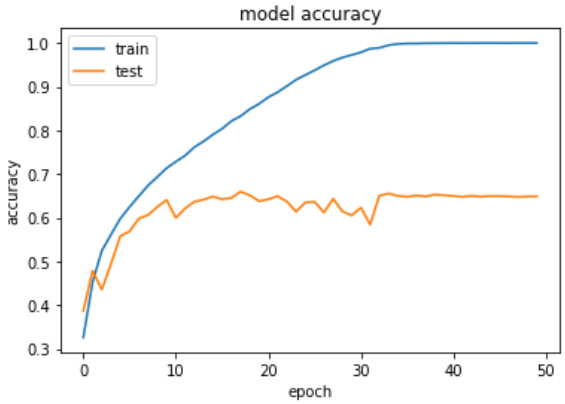
- προσθήκη dropout layers
- L1 - normalization στα βάρη
- L2 - normalization στα βάρη

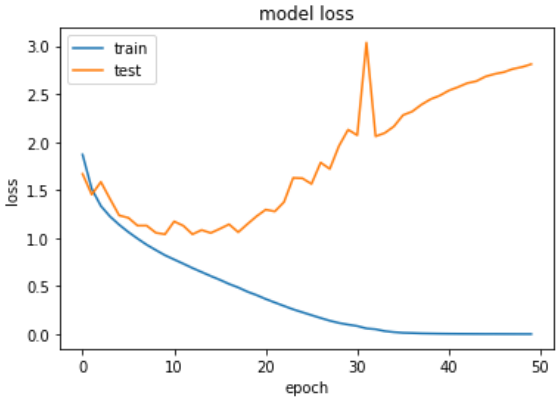
Στην τελευταία στήλη φαίνεται και ο χρόνος που έκανε το κάθε μοντέλο να εκπαιδευθεί.

characte-ristics information	Layer 1 neurons	Layer 2 neurons	Batch size	Learning rate	Dropout	L1 normali-zation	L2 normali-zation	Time (seconds)
1	64	128	64	0.01	-	-	-	445.5118
2	128	256	64	0.01	-	-	-	445.3157
3	128	256	64	0.01	0.2	-	-	445.7697
4	128	256	64	0.01	0.3	-	-	445.3898
5	128	256	64	0.01	0.5	-	-	416.1114
6	128	256	64	0.01	-	0.001	-	445.2427
7	128	256	64	0.01	-	0.01	-	445.5948
8	128	256	64	0.01	-	-	0.001	445.3090
9	128	256	64	0.01	-	-	0.01	445.2581
10	128	256	64	0.001	0.5	-	-	445.4835
11	128	256	64	0.1	0.5	-	-	423.6839
12	128	256	32	0.01	0.5	-	-	707.1732
13	128	256	16	0.01	0.5	-	-	1358.3610

Σαν πρώτο πείραμα, επιλέχθηκε ο αριθμός των νευρώνων του 1ου hidden layer να είναι ίσος με 64, και του δεύτερου hidden layer με 128. Σύνηθες είναι ο αριθμός των νευρώνων να διαιρείται επακριβώς με την είσοδο (εδώ 8192 μετά το flatten), και αξίζει να σημειωθεί ότι ξεκινήσαμε από έναν σχετικά μεγάλο αριθμό λόγω του μεγάλου πλήθους των στοιχείων του dataset. Το learning rate, εκτός από κάποια παραδείγματα στη συνέχεια, παρέμεινε στο 0.01, μία κλασσική τιμή για τέτοιου είδους προβλήματα. Παρακάτω παρουσιάζονται τα διαγράμματα της απόδοσης και του σφάλματος, το summary του μοντέλου, και το classification report που δείχνει τις μετρικές για κάθε κλάση ξεχωριστά:

1.





Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 32, 32, 32)	896
conv2d_3 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_1 (MaxPooling 2D)	(None, 16, 16, 32)	0
flatten_1 (Flatten)	(None, 8192)	0
dense_3 (Dense)	(None, 64)	524352
dense_4 (Dense)	(None, 128)	8320
dense_5 (Dense)	(None, 10)	1290

Total params: 544,106
Trainable params: 544,106
Non-trainable params: 0

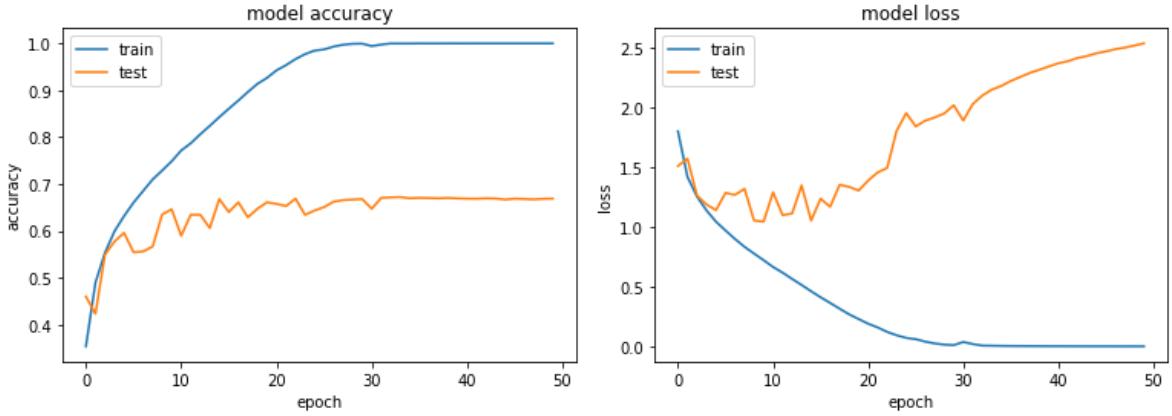
	precision	recall	f1-score	support
airplane	0.68	0.72	0.70	1000
automobile	0.77	0.74	0.76	1000
bird	0.55	0.52	0.53	1000
cat	0.45	0.47	0.46	1000
deer	0.60	0.56	0.58	1000
dog	0.54	0.53	0.53	1000
frog	0.69	0.73	0.71	1000
horse	0.70	0.71	0.70	1000
ship	0.79	0.78	0.78	1000
truck	0.73	0.73	0.73	1000
accuracy			0.65	10000
macro avg	0.65	0.65	0.65	10000
weighted avg	0.65	0.65	0.65	10000

Η τιμή του accuracy (0.65) είναι σχετικά καλή/αναμενόμενη για το μοντέλο μας,

καθώς το convolutional κομμάτι του είναι μικρό (με περισσότερα VGG blocks θα μπορούσε να αυξηθεί). Στο διάγραμμά μας όμως φαίνεται πως το loss αυξάνεται έντονα όσο περνούν τα epochs, κάτι το οποίο προβάλλει αναμφισβήτητα overfitting, το μοντέλο δηλαδή «μαθαίνει» υπερβολικά καλά το set εκπαίδευσης και αδυνατεί στη συνέχεια να εφαρμόσει αυτά που έμαθε σε νέα δεδομένα. Τρόποι αντιμετώπισης του φαινομένου υπάρχουν και αναλύονται στη συνέχεια.

Παρακάτω παρουσιάζονται οι αντίστοιχες πληροφορίες αλλά με αριθμό νευρώνων του πρώτου layer ίσο με 128 και του δεύτερου layer ίσο με 256. Θεωρητικά το αποτέλεσμα θα έπρεπε να είναι καλύτερο, διότι μπορεί να αυξηθεί η ακρίβεια με την πολυπλοκότητα (αλλά πάντα εξαρτάται από το πρόβλημα που έχουμε, οπότε προχωράμε στο πείραμα όπως και να ‘χει).

2.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 128)	1048704
dense_1 (Dense)	(None, 256)	33024
dense_2 (Dense)	(None, 10)	2570

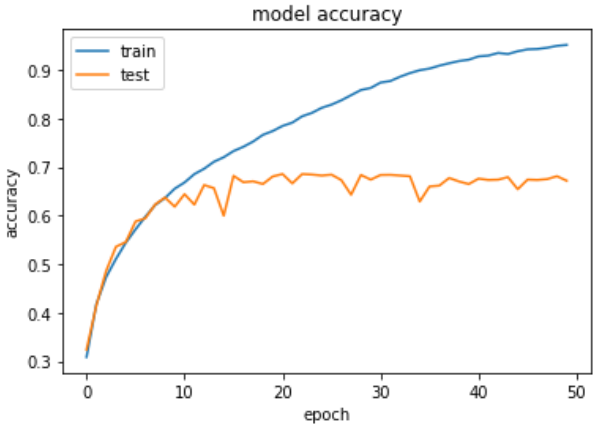
Total params: 1,094,442
Trainable params: 1,094,442
Non-trainable params: 0

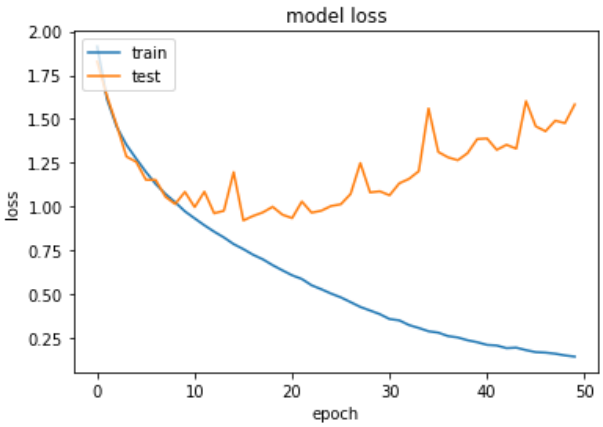
	precision	recall	f1-score	support
airplane	0.69	0.74	0.71	1000
automobile	0.77	0.76	0.77	1000
bird	0.56	0.54	0.55	1000
cat	0.48	0.47	0.48	1000
deer	0.62	0.62	0.62	1000
dog	0.57	0.58	0.57	1000
frog	0.72	0.76	0.74	1000
horse	0.73	0.73	0.73	1000
ship	0.80	0.77	0.79	1000
truck	0.74	0.73	0.73	1000
accuracy			0.67	10000
macro avg	0.67	0.67	0.67	10000
weighted avg	0.67	0.67	0.67	10000

Το accuracy φαίνεται να αυξήθηκε, αν και συνεχίζουμε να παρατηρούμε πρόβλημα overfitting στο διάγραμμα σφάλματος. Για τους παραπάνω λόγους, αποφασίσθηκε να κρατηθεί ο μεγαλύτερο αριθμός νευρώνων (128 & 256), και να δοκιμασθούν στα επόμενα πειράματα διάφορες τεχνικές αντιμετώπισης του overfitting, όπως η προσθήκη dropout layers και η προσθήκη L1-regularization και L2-regularization στο update των συναπτικών βαρών κάθε layer.

Πρώτη δοκιμή έγινε με την προσθήκη ενός dropout layer μετά από κάθε hidden layer. Περιληπτικά, η πιθανότητα που εισάγεται μέσα στο dropout layer δείχνει την πιθανότητα που έχει ένας νευρώνας να γίνει 0 (δηλαδή να αγνοηθεί) κατά τη διαδικασία της εκπαίδευσης. Η πρακτική αυτή βοηθά στην απλούστευση του μοντέλου μας και στην αποφυγή της υπερεκπαίδευσης (2 πράγματα άρρηκτα συνδεδεμένα). Όταν προχωρήσουμε όμως στην διαδικασία του testing θα συμμετέχουν κανονικά όλοι οι νευρώνες. Τα αποτελέσματα για dropout = 0.2, 0.3 & 0.5 παρουσιάζονται παρακάτω:

3.



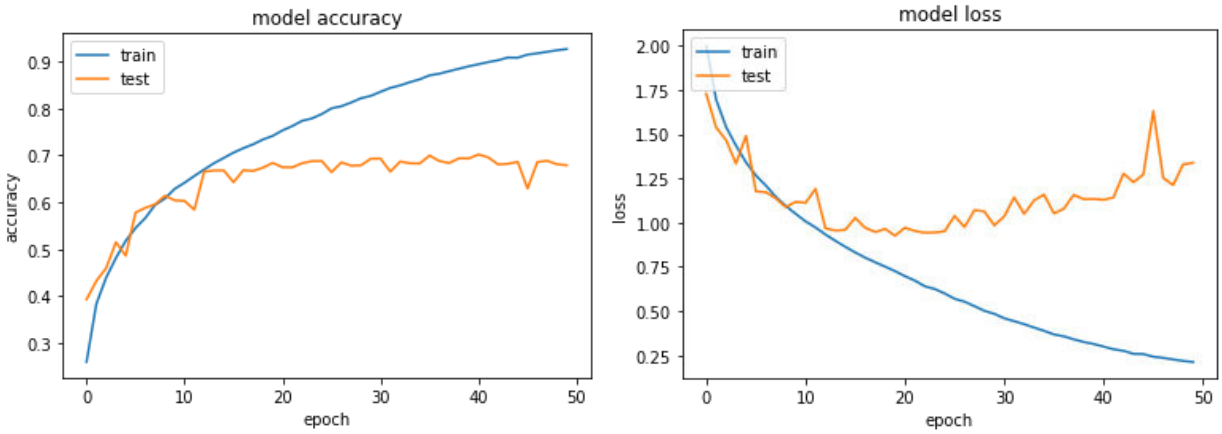


Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 32, 32, 32)	896
conv2d_3 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_1 (MaxPooling 2D)	(None, 16, 16, 32)	0
flatten_1 (Flatten)	(None, 8192)	0
dense_3 (Dense)	(None, 128)	1048704
dropout (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 256)	33024
dropout_1 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 10)	2570

Total params: 1,094,442
Trainable params: 1,094,442
Non-trainable params: 0

	precision	recall	f1-score	support
airplane	0.69	0.75	0.72	1000
automobile	0.82	0.76	0.79	1000
bird	0.52	0.62	0.56	1000
cat	0.45	0.57	0.50	1000
deer	0.74	0.45	0.56	1000
dog	0.55	0.60	0.57	1000
frog	0.73	0.76	0.75	1000
horse	0.86	0.62	0.72	1000
ship	0.82	0.79	0.80	1000
truck	0.73	0.80	0.76	1000
accuracy			0.67	10000
macro avg	0.69	0.67	0.67	10000
weighted avg	0.69	0.67	0.67	10000

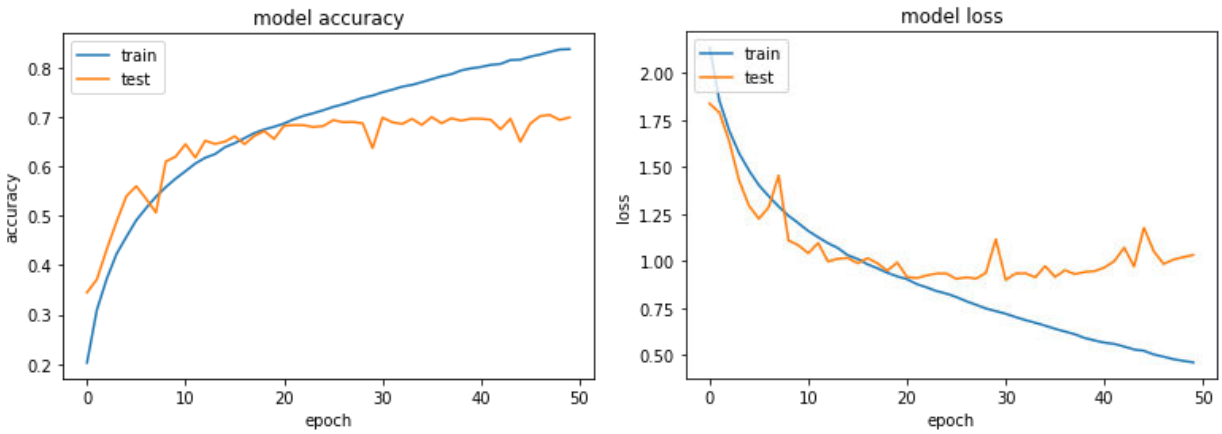
4.




```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)            (None, 32, 32, 32)        896

 conv2d_9 (Conv2D)            (None, 32, 32, 32)        9248

 max_pooling2d_4 (MaxPooling  (None, 16, 16, 32)        0
 2D)

 flatten_4 (Flatten)          (None, 8192)              0

 dense_12 (Dense)             (None, 128)               1048704

 dropout_2 (Dropout)          (None, 128)               0

 dense_13 (Dense)             (None, 256)               33024

 dropout_3 (Dropout)          (None, 256)               0

 dense_14 (Dense)             (None, 10)                2570

=================================================================
Total params: 1,094,442
Trainable params: 1,094,442
Non-trainable params: 0
_________________________________________________________________
```

	precision	recall	f1-score	support
airplane	0.71	0.72	0.71	1000
automobile	0.79	0.78	0.79	1000
bird	0.63	0.46	0.53	1000
cat	0.55	0.47	0.50	1000
deer	0.52	0.75	0.62	1000
dog	0.61	0.57	0.59	1000
frog	0.81	0.72	0.76	1000
horse	0.75	0.70	0.73	1000
ship	0.70	0.85	0.77	1000
truck	0.76	0.76	0.76	1000
accuracy			0.68	10000
macro avg	0.68	0.68	0.68	10000
weighted avg	0.68	0.68	0.68	10000

5.



Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 32, 32, 32)	896
conv2d_11 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_5 (MaxPooling 2D)	(None, 16, 16, 32)	0
flatten_5 (Flatten)	(None, 8192)	0
dense_15 (Dense)	(None, 128)	1048704
dropout_4 (Dropout)	(None, 128)	0
dense_16 (Dense)	(None, 256)	33024
dropout_5 (Dropout)	(None, 256)	0
dense_17 (Dense)	(None, 10)	2570

	precision	recall	f1-score	support
airplane	0.78	0.70	0.74	1000
automobile	0.81	0.84	0.82	1000
bird	0.62	0.54	0.58	1000
cat	0.55	0.42	0.48	1000
deer	0.57	0.68	0.62	1000
dog	0.61	0.63	0.62	1000
frog	0.67	0.85	0.75	1000
horse	0.79	0.73	0.76	1000
ship	0.82	0.81	0.81	1000
truck	0.77	0.79	0.78	1000
accuracy			0.70	10000
macro avg	0.70	0.70	0.70	10000
weighted avg	0.70	0.70	0.70	10000

Παρατηρούμε πως όσο αυξάνεται η πιθανότητα dropout p, τόσο αυξάνεται και το accuracy, το οποίο αγγίζει το 70% όταν p = 0.5. Αυτή η παρατήρηση, σε συνδυασμό με την ακόμα σημαντικότερη σταθεροποίηση του loss, μας οδηγεί στο συμπέρασμα πως αυτή η μέθοδος regularization λειτουργεί καλά με το πρόβλημα και το dataset που επιλέξαμε, οπότε θα μελετηθεί περαιτέρω στη συνέχεια. Ειδικά όταν p = 0.5, δεν παρατηρείται καθόλου overfitting στο διάγραμμα σφάλματος, πράγμα πολύ ενθαρρυντικό για το βέλτιστο μοντέλο στο οποίο θέλουμε να καταλήξουμε.

Στη συνέχεια, η μέθοδοι αντιμετώπισης του overfitting που μελετήθηκαν είναι οι L1 και L2 normalization. Οι μέθοδοι αυτοί ουσιαστικά εισάγουν ένα «penalty» στις τιμές των συναπτικών βαρών κατά τη διάρκεια της εκπαίδευσης (στην ουσία «τιμωρούνται» τα μεγάλα βάρη γιατί θέλουμε απλούστερο μοντέλο). Ανάλογα με το λ που επιλέγουμε αυξάνεται, λοιπόν, το σφάλμα του μοντέλου μας.

Πιο συγκεκριμένα, στην L1 κανονικοποίηση προστίθεται στο σφάλμα το άθροισμα της απόλυτης τιμής των βαρών πολλαπλασιασμένη με τον παράγοντα λ, ενώ στην L2 κανονικοποίηση προστίθεται το άθροισμα των τετραγώνων των βαρών πολλαπλασιασμένο (ομοίως) με τον παράγοντα λ. Η «τιμωρία» αυτή των μεγάλων βαρών οδηγεί σε μεγάλα σφάλματα, τα οποία με τη σειρά τους κατά τη διάρκεια της εκπαίδευσης οδηγούν σε σμίκρυνση των συναπτικών βαρών (ιδανικά προς το 0).

Αν ένα βάρος έχει μεγάλη (απόλυτη) τιμή, η L1 κανονικοποίηση μικραίνει το βάρος αυτό πολύ λιγότερο από την L2. Από την άλλη, αν το βάρος w είναι μικρό, η L1 κανονικοποίηση το επηρεάζει πολύ περισσότερο από την L2. Σημειώνεται επίσης ότι η L1 normalization μειώνει τα βάρη σταθερά προς το μηδέν (μιας και χρησιμοποιεί την απόλυτη τιμή), ενώ η L2 normalization μειώνει τα βάρη με ρυθμό που εξαρτάται από την τιμή του κάθε βάρους (ένα πολύ μεγάλο βάρος οδηγεί σε πολύ μεγαλύτερη/γρηγορότερη μείωση, μιας και προστίθεται στο σφάλμα το τετράγωνο του).

$$C = C_0 + \frac{\lambda}{2n}\sum_w w^2$$

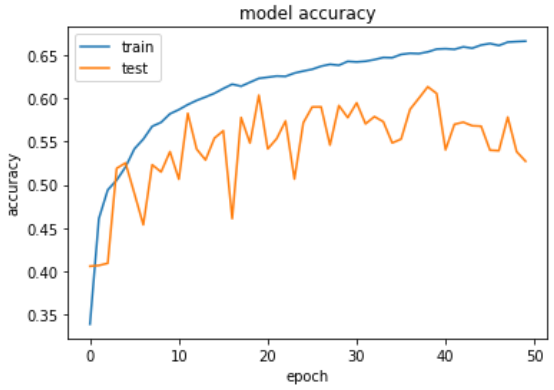
L2 – NORMALIZATION

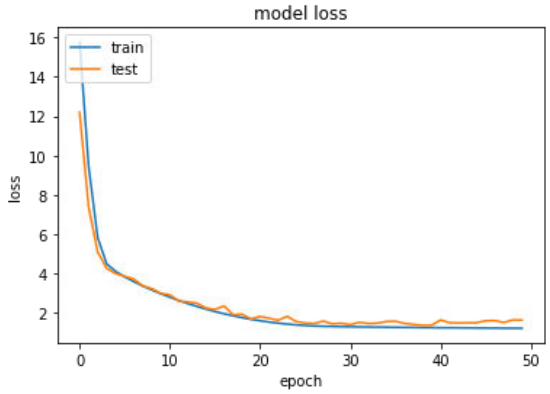
$$C = C_0 + \frac{\lambda}{n}\sum_w |w|$$

L1 - NORMALIZATION

Παρακάτω παρουσιάζονται διαγράμματα και πληροφορίες για λ = 0.001, 0.01 στις L1 & L2 normalization αντίστοιχα.

6.



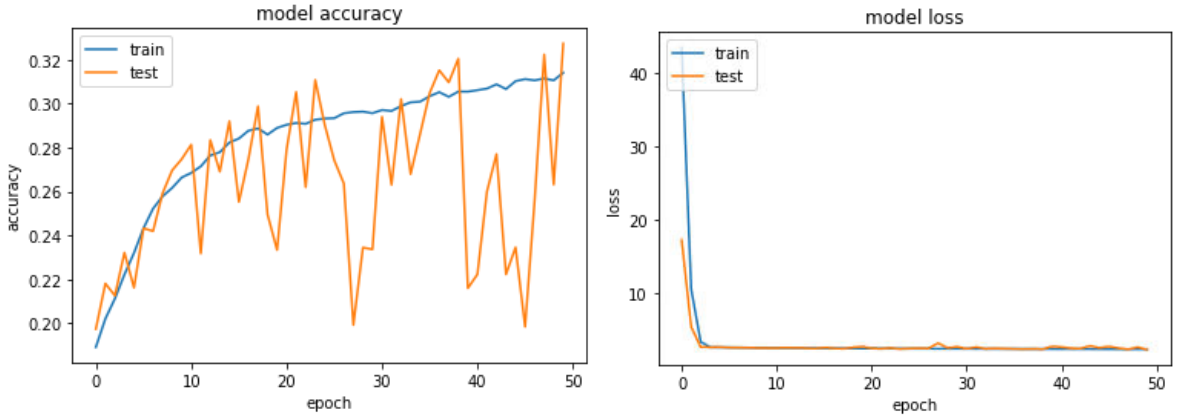


Layer (type)	Output Shape	Param #
conv2d_14 (Conv2D)	(None, 32, 32, 32)	896
conv2d_15 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_7 (MaxPooling 2D)	(None, 16, 16, 32)	0
flatten_7 (Flatten)	(None, 8192)	0
dense_21 (Dense)	(None, 128)	1048704
dense_22 (Dense)	(None, 256)	33024
dense_23 (Dense)	(None, 10)	2570

Total params: 1,094,442
Trainable params: 1,094,442
Non-trainable params: 0

	precision	recall	f1-score	support
airplane	0.58	0.53	0.56	1000
automobile	0.76	0.74	0.75	1000
bird	0.30	0.77	0.43	1000
cat	0.54	0.12	0.20	1000
deer	0.67	0.32	0.43	1000
dog	0.36	0.72	0.48	1000
frog	0.89	0.44	0.59	1000
horse	0.56	0.74	0.64	1000
ship	0.95	0.37	0.53	1000
truck	0.82	0.51	0.63	1000
accuracy			0.53	10000
macro avg	0.64	0.53	0.52	10000
weighted avg	0.64	0.53	0.52	10000

7.

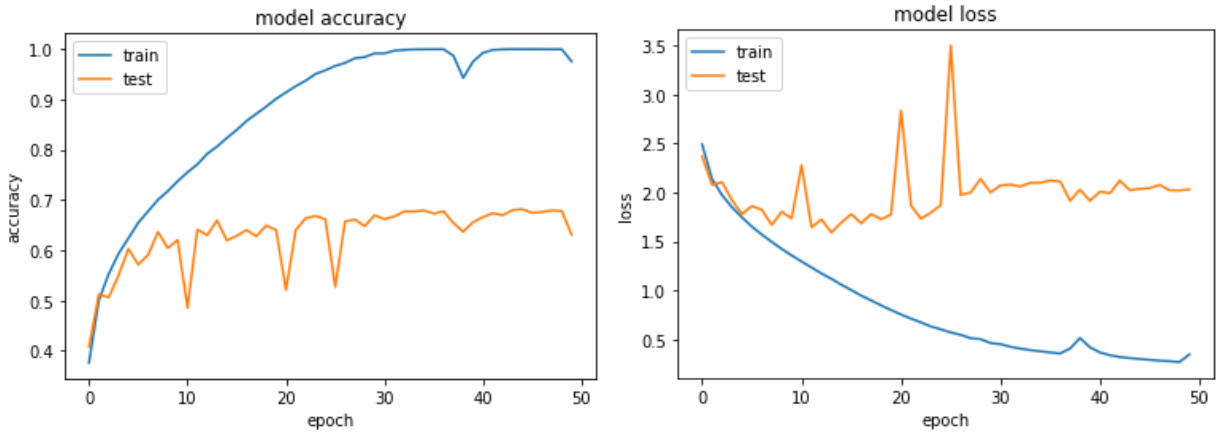


Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 32, 32, 32)	896
conv2d_13 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_6 (MaxPooling 2D)	(None, 16, 16, 32)	0
flatten_6 (Flatten)	(None, 8192)	0
dense_18 (Dense)	(None, 128)	1048704
dense_19 (Dense)	(None, 256)	33024
dense_20 (Dense)	(None, 10)	2570

Total params: 1,094,442
Trainable params: 1,094,442
Non-trainable params: 0

	precision	recall	f1-score	support
airplane	0.54	0.35	0.43	1000
automobile	0.46	0.59	0.52	1000
bird	0.26	0.16	0.20	1000
cat	0.18	0.10	0.13	1000
deer	0.30	0.30	0.30	1000
dog	0.21	0.04	0.07	1000
frog	0.27	0.67	0.38	1000
horse	0.23	0.20	0.21	1000
ship	0.49	0.53	0.51	1000
truck	0.28	0.33	0.30	1000
accuracy			0.33	10000
macro avg	0.32	0.33	0.30	10000
weighted avg	0.32	0.33	0.30	10000

8.

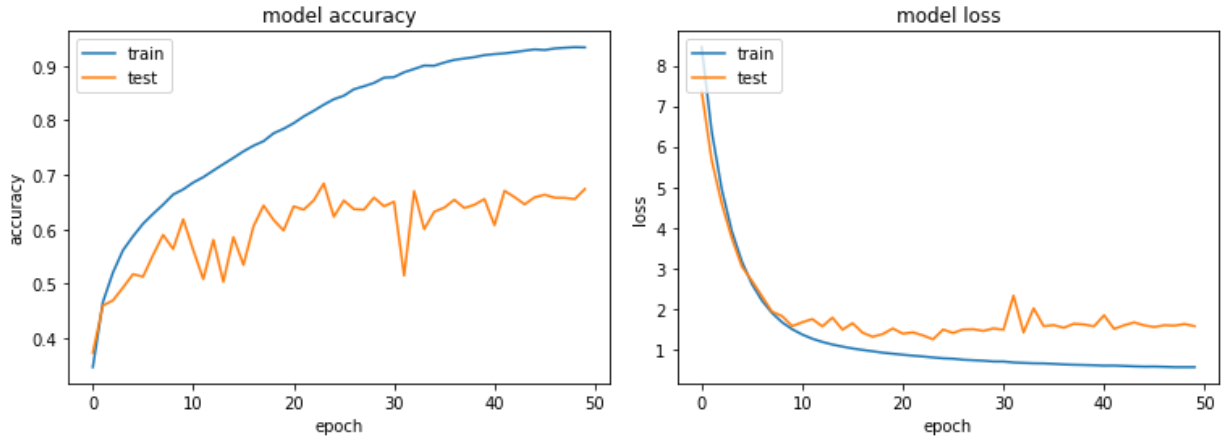


Layer (type)	Output Shape	Param #
conv2d_16 (Conv2D)	(None, 32, 32, 32)	896
conv2d_17 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_8 (MaxPooling 2D)	(None, 16, 16, 32)	0
flatten_8 (Flatten)	(None, 8192)	0
dense_24 (Dense)	(None, 128)	1048704
dense_25 (Dense)	(None, 256)	33024
dense_26 (Dense)	(None, 10)	2570

Total params: 1,094,442
Trainable params: 1,094,442
Non-trainable params: 0

	precision	recall	f1-score	support
airplane	0.76	0.64	0.69	1000
automobile	0.83	0.68	0.75	1000
bird	0.36	0.74	0.49	1000
cat	0.52	0.35	0.42	1000
deer	0.62	0.53	0.57	1000
dog	0.53	0.51	0.52	1000
frog	0.69	0.78	0.73	1000
horse	0.75	0.67	0.71	1000
ship	0.82	0.70	0.76	1000
truck	0.76	0.71	0.74	1000
accuracy			0.63	10000
macro avg	0.66	0.63	0.64	10000
weighted avg	0.66	0.63	0.64	10000

9.



Layer (type)	Output Shape	Param #
conv2d_18 (Conv2D)	(None, 32, 32, 32)	896
conv2d_19 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_9 (MaxPooling 2D)	(None, 16, 16, 32)	0
flatten_9 (Flatten)	(None, 8192)	0
dense_27 (Dense)	(None, 128)	1048704
dense_28 (Dense)	(None, 256)	33024
dense_29 (Dense)	(None, 10)	2570

	precision	recall	f1-score	support
airplane	0.78	0.67	0.72	1000
automobile	0.78	0.78	0.78	1000
bird	0.71	0.42	0.53	1000
cat	0.48	0.54	0.51	1000
deer	0.56	0.70	0.62	1000
dog	0.65	0.46	0.53	1000
frog	0.76	0.75	0.75	1000
horse	0.64	0.82	0.72	1000
ship	0.81	0.78	0.79	1000
truck	0.66	0.83	0.74	1000
accuracy			0.67	10000
macro avg	0.68	0.67	0.67	10000
weighted avg	0.68	0.67	0.67	10000

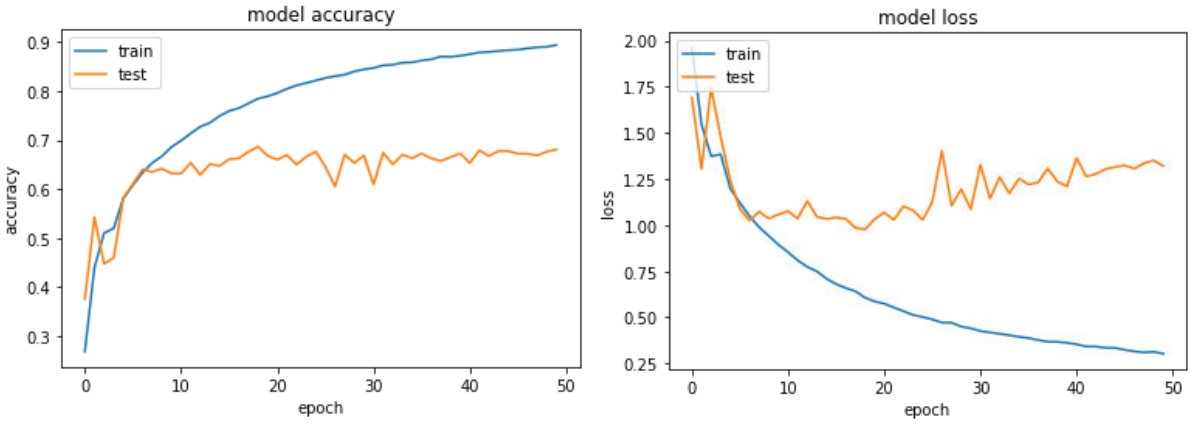
Παρατηρείται πρώτα από όλα, πως η L1 κανονικοποίηση δεν έχει καθόλου καλά αποτελέσματα, με accuracy 0.53 και 0.33 όταν λ=0.001 και λ=0.01 αντίστοιχα. Η απόδοση αυτή είναι χειρότερη από το αρχικό, χωρίς εξτρά κανονικοποιήσεις, μοντέλο, οπότε η L1 normalization, αν και όντως οδηγεί στην αποφυγή της υπερεκπαίδευσης, δεν προσφέρει χρήσιμα αποτελέσματα σε καμία περίπτωση.

Όσον αφορά το L2 normalization, οι αποδόσεις είναι αρκετά καλύτερες (0.63 & 0.67) και αποφεύγεται επιτυχημένα το overfitting. Το loss στο οποίο τελικά σταθεροποιείται το μοντέλο, όμως, κυμαίνεται γύρω στο 2, σε αντίθεση το τελικό loss στο dropout μοντέλο, το οποίο παίρνει τιμές γύρω από το 1. Τελικά δηλαδή, αν και σχετικά καλό μοντέλο, δεν υπερτερεί σε σχέση με την μέθοδο προσθήκης dropout layer για το συγκεκριμένο πρόβλημα.

Μετά το πέρας των πρώτων ενδεικτικών πειραμάτων, και αφού καταλήξαμε στο καλύτερο προς το παρόν μοντέλο με προσθήκη 2 dropout layers που έχουν πιθανότητα για dropping out ίση με 0.5, προχωρήσαμε σε μερικά ακόμη πειράματα με τα παραπάνω σαν βάση, σε συνδυασμό με αλλαγή κάποιων άλλων, πριν σταθερών, παραμέτρων.

Έτσι, προχωρήσαμε σε 2 πειράματα όπου αλλάξαμε τον -προηγουμένως σταθερό ρυθμό εκμάθησης (learning rate) του optimizer. Ο ρυθμός εκμάθησης δείχνει, ουσιαστικά, πόσο πολύ ή λίγο θα κάνουν update τα συναπτικά βάρη μετά από κάθε επανάληψη. Ένας μεγάλος ρυθμός εκμάθησης, για παράδειγμα, βοηθά το μοντέλο να συγκλίνει πολύ γρηγορότερα, μπορεί όμως η σύγκλιση αυτή να μην είναι η καλύτερη δυνατή. Από την άλλη, ένας μικρός ρυθμός εκμάθησης είναι σχετικά αποδεδειγμένο πως θα μας δώσει καλύτερα αποτελέσματα, όμως μπορεί να χρειαστεί πολύ περισσότερο χρόνο/epochs για να το κάνει αυτό, resources που πιθανό να μη μπορούμε να διαθέσουμε, ειδικά όσον αφορά πολύ μεγάλα datasets ή πολύ μικρά batch sizes. Στην συνέχεια παρουσιάζονται τα διαγράμματα απόδοσης και σφάλματος, και οι πληροφορίες για lr = 0.1 και lr = 0.001:

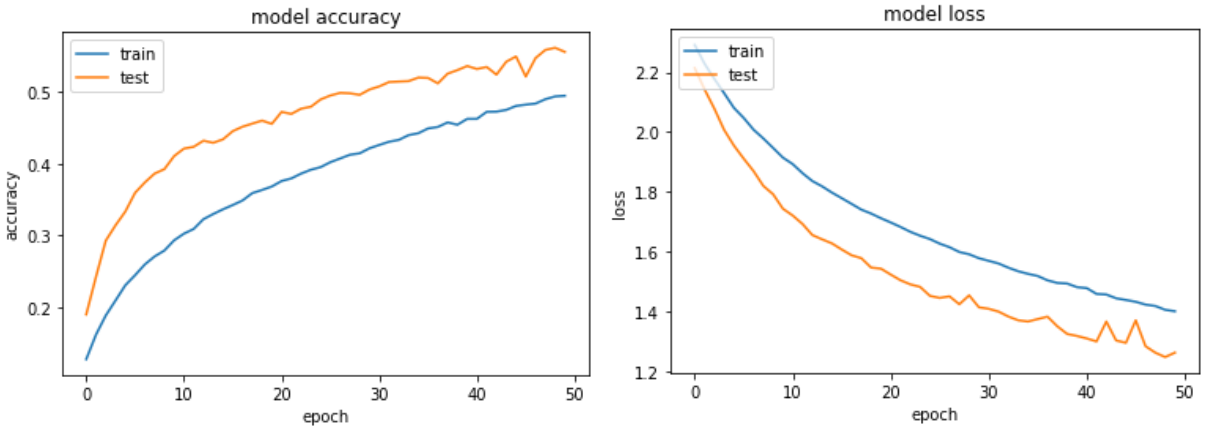
10.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 128)	1048704
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 256)	33024
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 10)	2570

	precision	recall	f1-score	support
airplane	0.73	0.74	0.73	1000
automobile	0.79	0.82	0.81	1000
bird	0.59	0.50	0.54	1000
cat	0.44	0.54	0.48	1000
deer	0.61	0.64	0.63	1000
dog	0.58	0.58	0.58	1000
frog	0.78	0.72	0.75	1000
horse	0.75	0.71	0.73	1000
ship	0.78	0.81	0.80	1000
truck	0.80	0.75	0.77	1000
accuracy			0.68	10000
macro avg	0.69	0.68	0.68	10000
weighted avg	0.69	0.68	0.68	10000

11.



Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 32, 32, 32)	896
conv2d_3 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_1 (MaxPooling 2D)	(None, 16, 16, 32)	0
flatten_1 (Flatten)	(None, 8192)	0
dense_3 (Dense)	(None, 128)	1048704
dropout_2 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 256)	33024
dropout_3 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 10)	2570

	precision	recall	f1-score	support
airplane	0.58	0.65	0.61	1000
automobile	0.67	0.75	0.71	1000
bird	0.41	0.31	0.36	1000
cat	0.37	0.35	0.36	1000
deer	0.47	0.44	0.45	1000
dog	0.42	0.60	0.50	1000
frog	0.63	0.62	0.62	1000
horse	0.61	0.68	0.64	1000
ship	0.74	0.61	0.67	1000
truck	0.71	0.54	0.61	1000
accuracy			0.56	10000
macro avg	0.56	0.56	0.55	10000
weighted avg	0.56	0.56	0.55	10000

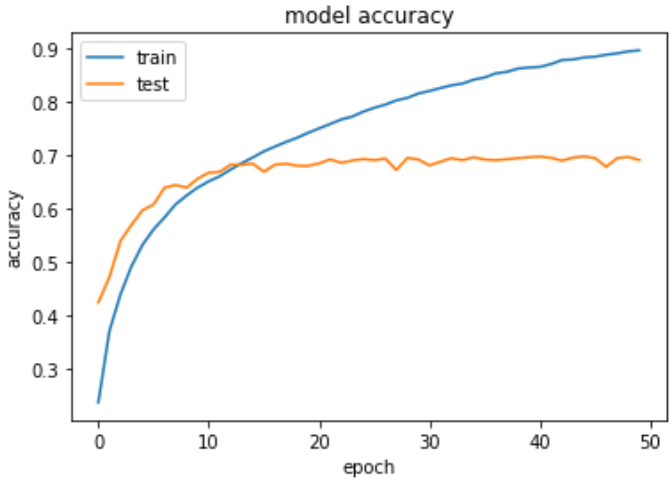
Η απόδοση του 1ου μοντέλου (lr = 0.1) δεν παρουσιάζει πολλές διαφορές σε σχέση με το αρχικό μας μοντέλο (lr = 0.01), είναι λίγο μικρότερη (0.68) αλλά τα διαγράμματα τους είναι πολύ παρόμοια. Ο ρυθμός εκμάθησης λοιπόν δεν παρουσιάζει μεγάλες διαφορές ανάμεσα στις 2 αυτές τιμές όταν εκπαιδεύουμε σε 50 epochs, αν και η αρχική μας επιλογή ήταν καλύτερη.

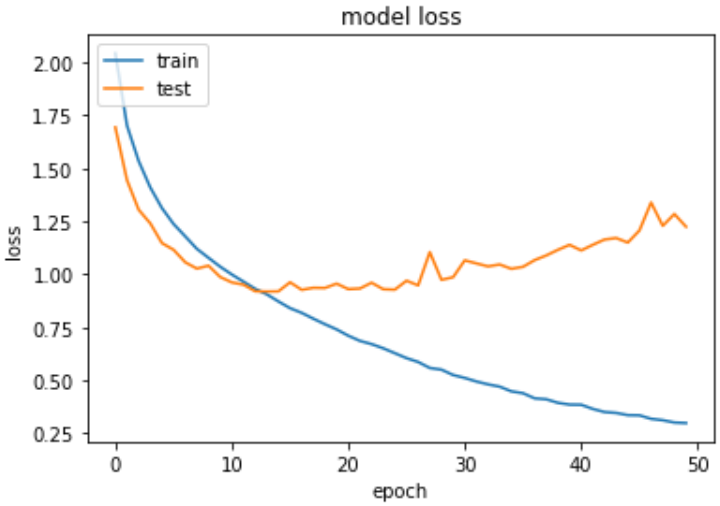
Όταν επιλέγουμε lr = 0.0001 όμως η απόδοση πέφτει αρκετά (0.56), πτώση αναμενόμενη, καθώς (όπως προαναφέρθηκε), ο μικρότερος ρυθμός εκμάθησης χρειάζεται και περισσότερο χρόνο εκπαίδευσης/περισσότερα epochs, πειράματα στα οποία δεν προχωρήσαμε.

Τέλος, ο παράγοντας εκπαίδευσης που αλλάξαμε στην εκπαίδευση μας, αναζητώντας καλύτερη απόδοση, είναι το batch size. Με λίγα λόγια, το batch size δείχνει πόσα δεδομένα/εισόδους θα περάσει το μοντέλο κατά τη διαδικασία του training μέχρι να κάνει update τα συναπτικά βάρη σύμφωνα με την αντικειμενική συνάρτηση που έχει επιλεχθεί. Θεωρητικά, μικρότερα batch sizes παρουσιάζουν καλύτερα αποτελέσματα, ενέχουν όμως τον κίνδυνο της υπερεκπαίδευσης ξανά (συχνότερο update στα συναπτικά βάρη οδηγεί σε μοντέλο πιο «περίπλοκο», που δεν μπορεί να προβλέψει σωστά τα νέα δεδομένα).

Στη συνέχεια φαίνονται τα 2 τελευταία πειράματα που υλοποιήθηκαν, με batch size = 32 και batch size = 16:

12.

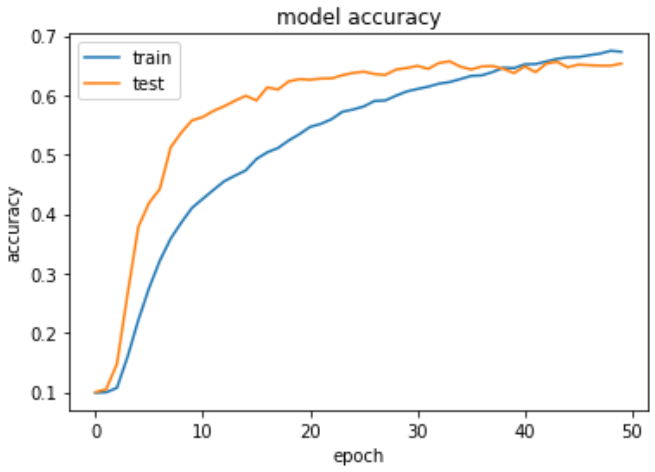


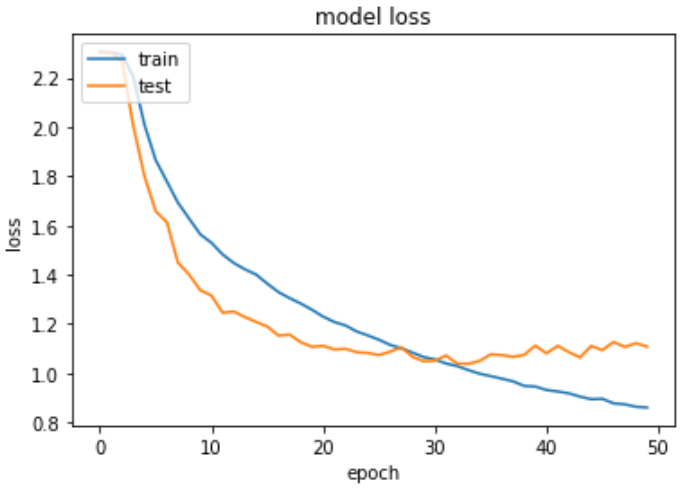


Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 32, 32, 32)	896
conv2d_7 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_3 (MaxPooling 2D)	(None, 16, 16, 32)	0
flatten_3 (Flatten)	(None, 8192)	0
dense_9 (Dense)	(None, 128)	1048704
dropout_6 (Dropout)	(None, 128)	0
dense_10 (Dense)	(None, 256)	33024
dropout_7 (Dropout)	(None, 256)	0
dense_11 (Dense)	(None, 10)	2570

	precision	recall	f1-score	support
airplane	0.77	0.70	0.73	1000
automobile	0.81	0.82	0.82	1000
bird	0.64	0.48	0.55	1000
cat	0.47	0.54	0.50	1000
deer	0.61	0.67	0.64	1000
dog	0.59	0.58	0.59	1000
frog	0.73	0.77	0.75	1000
horse	0.74	0.76	0.75	1000
ship	0.79	0.83	0.81	1000
truck	0.78	0.77	0.77	1000
accuracy			0.69	10000
macro avg	0.69	0.69	0.69	10000
weighted avg	0.69	0.69	0.69	10000

13.





Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 32)	262176
dropout (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 64)	2112
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 10)	650

Total params: 275,082
Trainable params: 275,082
Non-trainable params: 0

	precision	recall	f1-score	support
airplane	0.69	0.67	0.68	1000
automobile	0.70	0.78	0.74	1000
bird	0.56	0.51	0.53	1000
cat	0.44	0.49	0.46	1000
deer	0.60	0.61	0.60	1000
dog	0.64	0.54	0.59	1000
frog	0.72	0.76	0.74	1000
horse	0.77	0.70	0.73	1000
ship	0.73	0.77	0.75	1000
truck	0.71	0.70	0.70	1000
accuracy			0.65	10000
macro avg	0.65	0.65	0.65	10000
weighted avg	0.65	0.65	0.65	10000

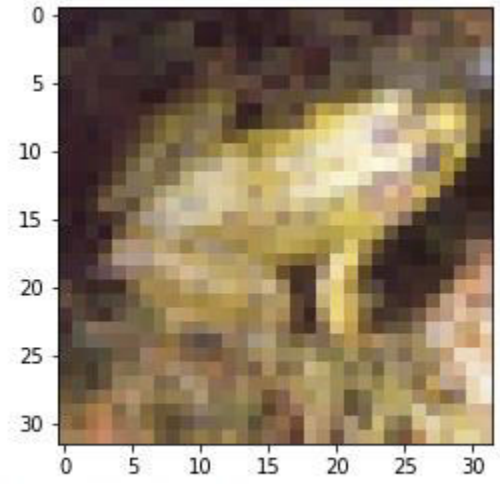
Παρατηρείται ότι στο batch size = 32 η απόδοση (accuracy = 0.69) είναι εξίσου καλή με το αρχικό μας (batch size =64) μοντέλο, ο χρόνος όμως είναι αρκετά μεγαλύτερος, οπότε δε θα επιλεγόταν για την εκπαίδευση στο συγκεκριμένο πρόβλημα, μιας και δεν προσφέρει κάτι παραπάνω . Το batch size = 16, από την άλλη, έχει μικρότερη απόδοση, αν και το loss διάγραμμα του μοντέλου δεν παρουσιάζει σημάδια υπερεκπαίδευσης, οπότε καταλήγουμε στο συμπέρασμα πως δεν είναι κατάλληλο για το πρόβλημα μας.

Σημειώνεται, τέλος, πως ο χρόνος εκπαίδευσης όλων των παραπάνω μοντέλων όταν batch size = 64 είναι πάνω κάτω ο ίδιος, ενώ όταν batch size = 32 & 16, ο χρόνος διπλασιάζεται και τετραπλασιάζεται αντίστοιχα (όπως είναι και αναμενόμενο).

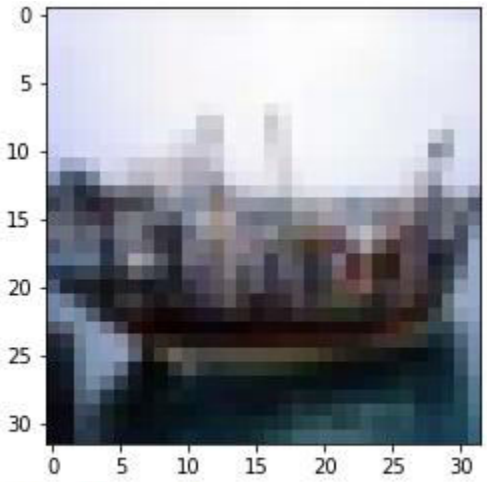
Ο σταθερός χρόνος με batch size = 64 (~445 seconds) δεν είναι ιδιαίτερα συγκρίσιμος ούτε με το k-nearest neighbor (~23 seconds) ούτε με το nearest centroid (~2 seconds) μοντέλο, αλλά η απόδοση είναι παραπάνω από διπλάσια, οπότε το πιθανότερο είναι πως -για αυτό το πρόβλημα- η θυσία του χρόνου θα ήταν επιθυμητή για την λήψη καλύτερων και ακριβέστερων αποτελεσμάτων.

Στο τελευταίο σημείο της αναφοράς παρουσιάζω μερικά παραδείγματα σωστής και λανθασμένης κατηγοριοποίησης των διαφόρων κλάσεων του dataset cifar-10, με χρήση του μοντέλου 5.

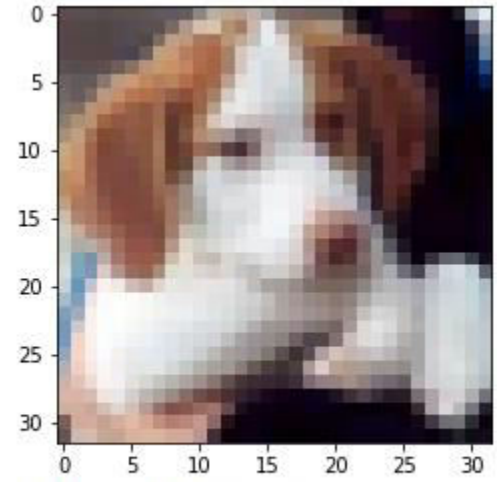
Σωστές προβλέψεις:



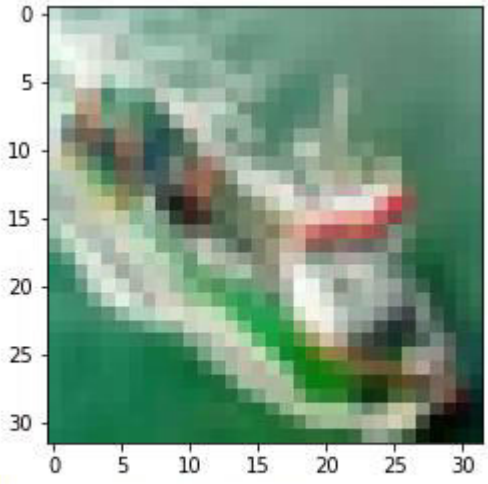
Model prediction: 6
frog



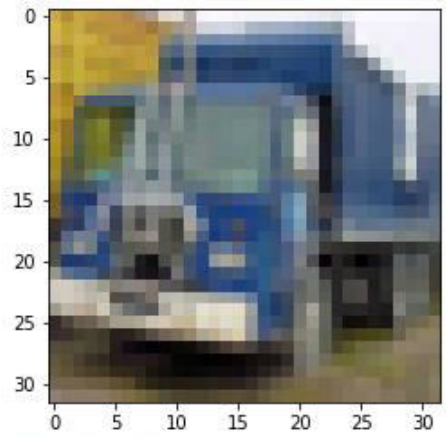
Model prediction: 8
ship



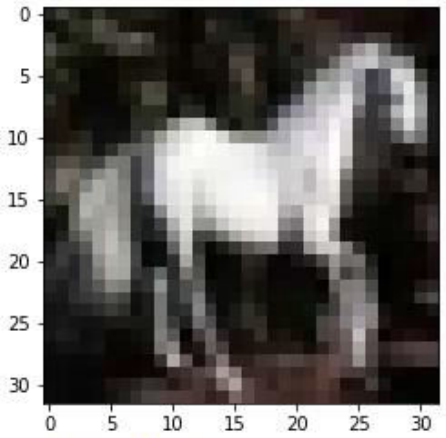
Model prediction: 5
dog



Model prediction: 8
ship

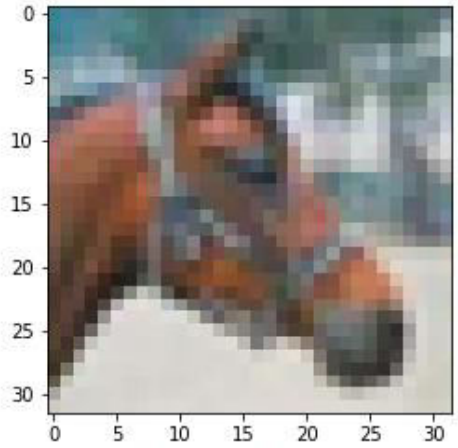


Model prediction: 9
truck

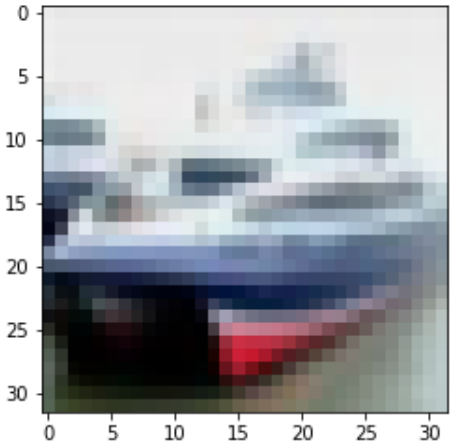


Model prediction: 7
horse

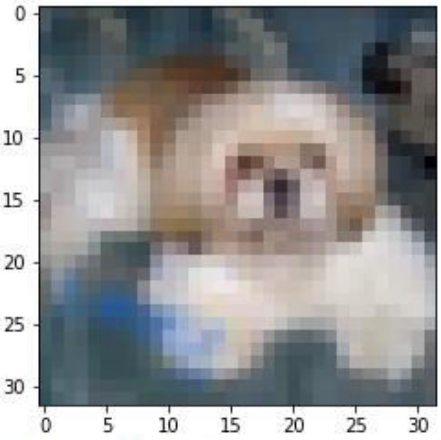
Λανθασμένες προβλέψεις:



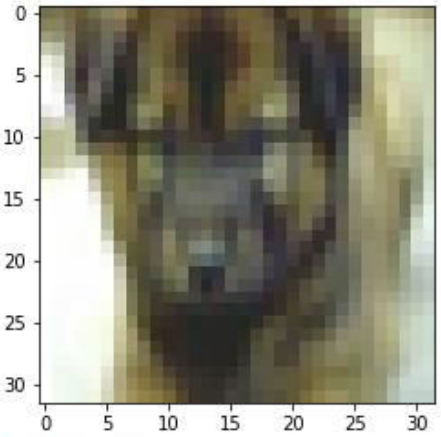
Model prediction: 3
cat



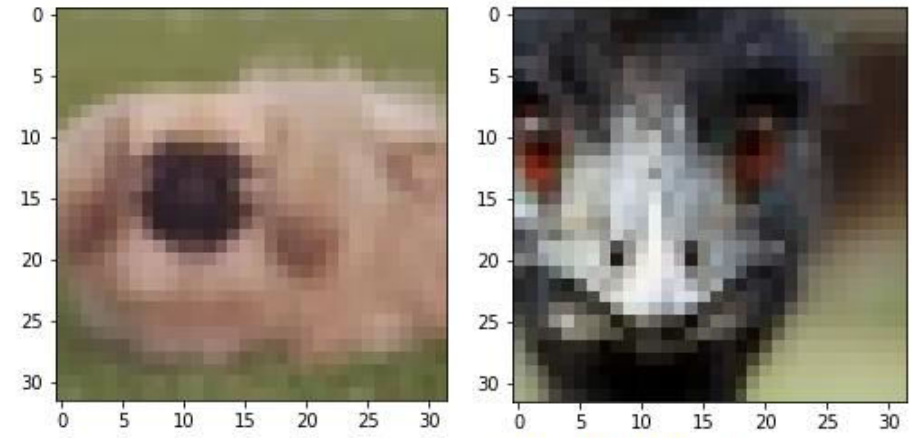
Model prediction: 1
automobile



Model prediction: 3
cat



Model prediction: 3
cat



ΒΙΒΛΙΟΓΡΑΦΙΑ

https://github.com/shreyagu/kNN_CiFAR10dataset

https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

https://arxiv.org/abs/1409.1556

https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

https://keras.io/api/layers/convolution_layers/convolution2d/

https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

https://github.com/eijaz1/k-NN-Tutorial-Scikit-learn/blob/master/KNN%20Tutorial.ipynb

https://machinelearningmastery.com/nearest-shrunken-centroids-with-python/

https://medium.com/@erikgreenj/k-neighbors-classifier-with-gridsearchcv-basics-3c445ddeb657

https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/

https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e

https://slideplayer.com/slide/17404709/