Νευρωνικά Δίκτυα - Βαθιά Μάθηση

1^{η} Υποχρεωτική Εργασία

Παπαδόπουλος Παναγιώτης 10697 ΗΜΜΥ

Σετ Δεδομένων (Dataset)

Για την ανάπτυξη ενός νευρωνικού δικτύου εμπρόσθιας τροφοδότησης (feedforward Neural Network) επιλέχθηκε συνδυασμός συνελικτικού (CNN) και πλήρως συνδεδεμένου δικτύου (MLP). Χρησιμοποιήθηκε η βάση δεδομένων tiny-imagenet, η οποία έχει 110.000 εικόνες 200 κλάσεων, οι οποίες χωρίζονται σε 100.000 training και 10.000 validation. Οι τελευταίες 10.000 χωρίστηκαν περαιτέρω σε 5.000 validation και 5.000 testing εικόνες. Ο αριθμός των εικόνων ανά κλάση στα δύο τελευταία σετ είναι ο ίδιος (25 εικόνες για κάθε κλάση). Αφού χωριστούν οι εικόνες στα τρία συνολικά σετ, αφαιρούνται οι εικόνες με κωδικοποίηση grayscale και οι τελικές εικόνες διαιρούνται με συντελεστή 255 για να έχει το νευρωνικό ως είσοδο τιμές εντός του εύρους $[0,1]$.

Εργαλεία

Για την υλοποίηση του νευρωνικού δικτύου επιλέχθηκε η γλώσσα προγραμματισμού $python$ και η βιβλιοθήκη $keras$ για την ανάπτυξή του. Άλλες βιβλιοθήκες που χρησιμοποιήθηκαν είναι οι εξής:

- $datasets$, για την εισαγωγή των εικόνων
- $matplotlib$, για την παρουσίαση διαγραμμάτων
- $numpy$, για τον εύκολο χειρισμό δεδομένων
- $time$, για την μέτρηση του χρόνου εκπαίδευσης
- $Image$, για την επεξεργασία εικόνων

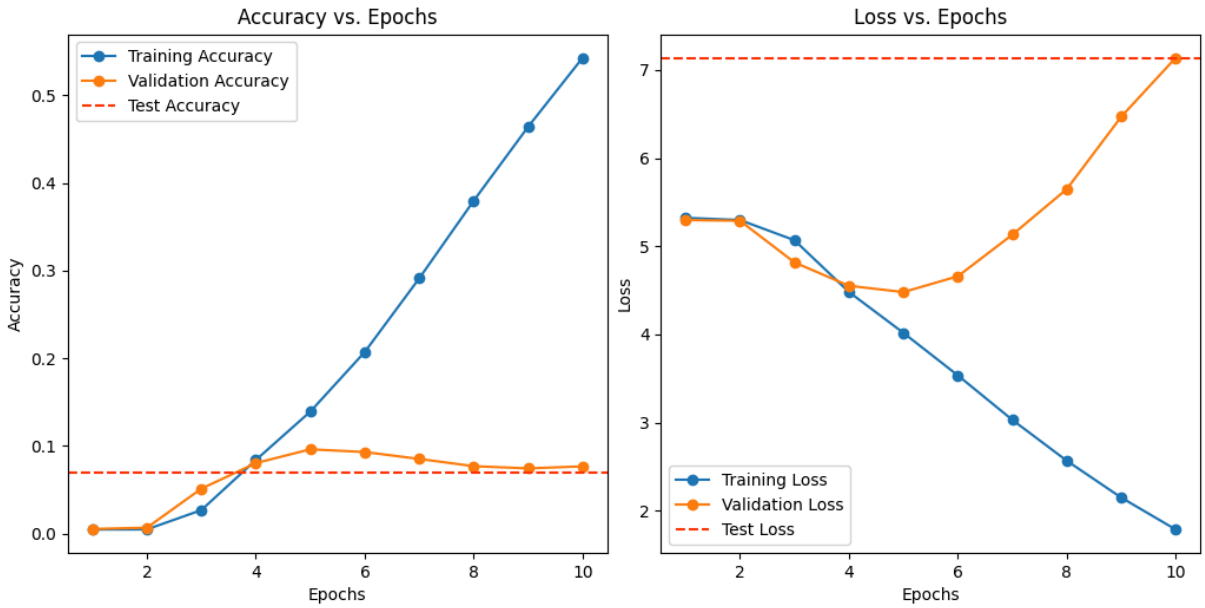
Δομή του Νευρωνικού Δικτύου

Το νευρωνικό δίκτυο που θα αναπτυχθεί θα αποτελείται από 3 ειδών στρώσεις (layers): συνελικτικές στρώσεις, max-pooling στρώσεις και πλήρως συνδεδεμένες (fully connected) στρώσεις. Ο στόχος είναι να βρεθεί η πιο αποτελεσματική αναλογία αυτών και των χαρακτηριστικών τους με σκοπό τη μεγιστοποίηση της ευστοχίας και την ελαχιστοποίηση της απώλειας στην κατηγοριοποίηση.

Ανάπτυξη του Νευρωνικού Δικτύου

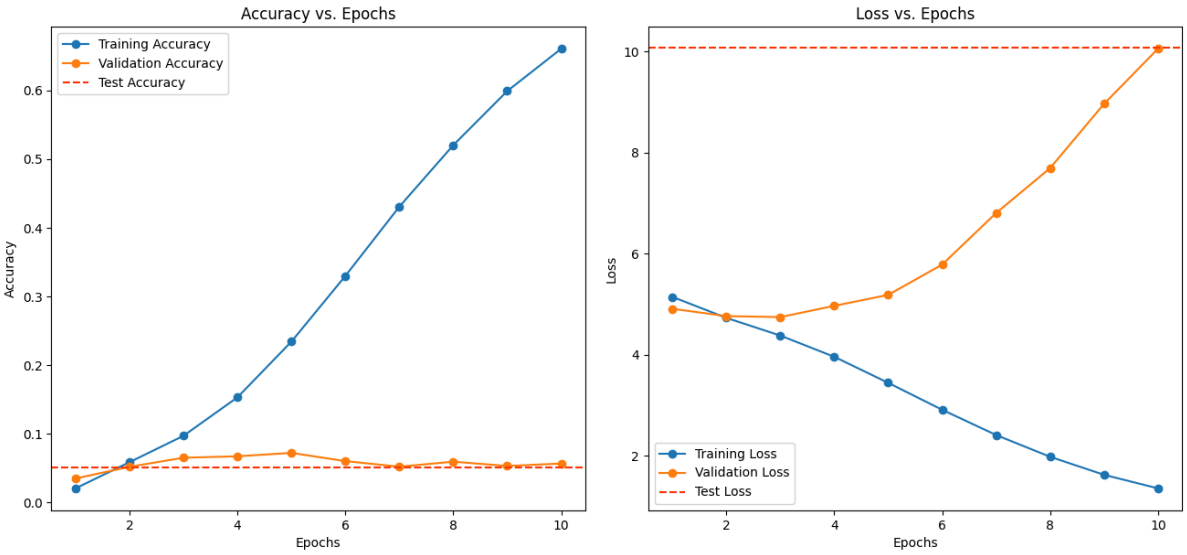
- **Συνελικτικές & Max-Pooling Στρώσεις**

Για την απόκτηση ενός επιπέδου αναφοράς, θα υλοποιηθεί ένα απλό μοντέλο ενός συνελικτικού layer με 32 φίλτρα, $3x3$ μέγεθος φίλτρων, 'έγκυρο' padding (valid padding) και stride $1x1$, ενός max-pooling layer με pool size $2x2$ και stride $1x1$, ενός πλήρως συνδεδεμένο layer με 128 νευρώνες με συνάρτηση ενεργοποίησης ReLU και ένα output layer με 200 νευρώνες και συνάρτηση ενεργοποίησης softmax, όπως συνηθίζεται στα νευρωνικά δίκτυα κατηγοριοποίησης. Θα χρησιμοποιηθεί ο optimizer Adam και η απώλεια θεωρείται η sparse categorical cross entropy loss. Μετά από 10 εποχές εκπαίδευσης, προκύπτουν τα παρακάτω αποτελέσματα.

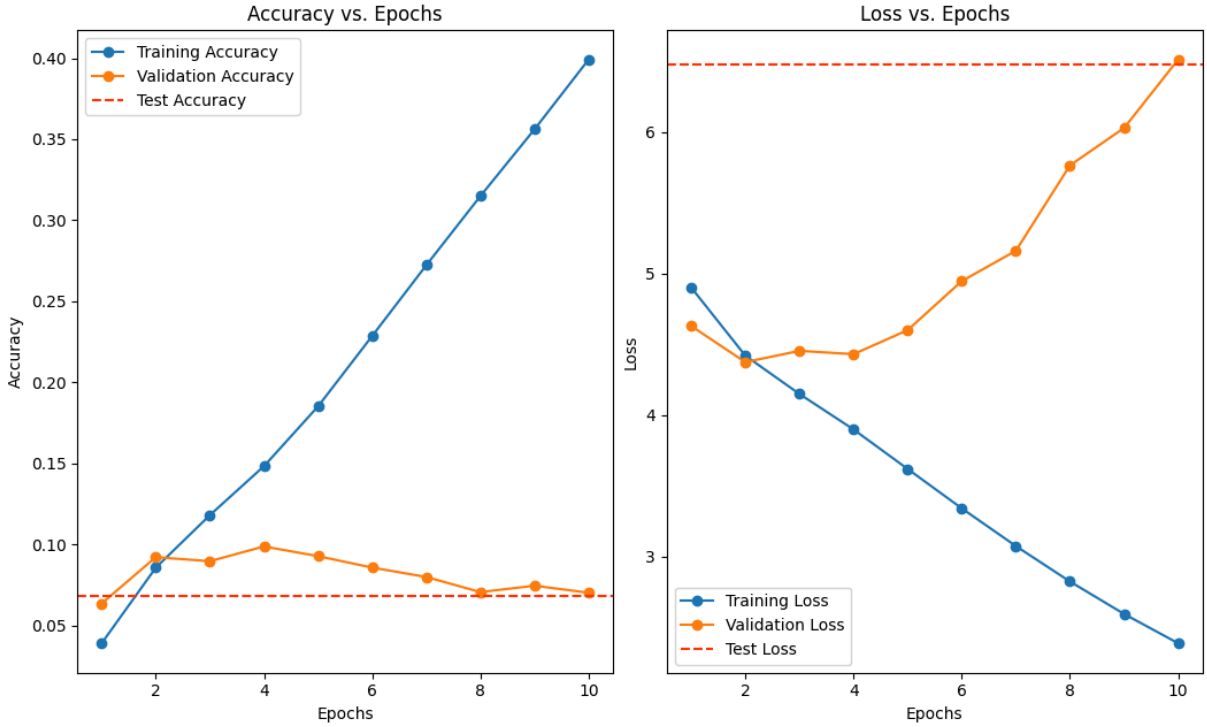


Ο χρόνος εκπαίδευσης ήταν 33 λεπτά. Αντίστοιχα, αλλάζοντας το μέγεθος των φίλτρων σε 5x5, 7x7 προκύπτουν τα παρακάτω.

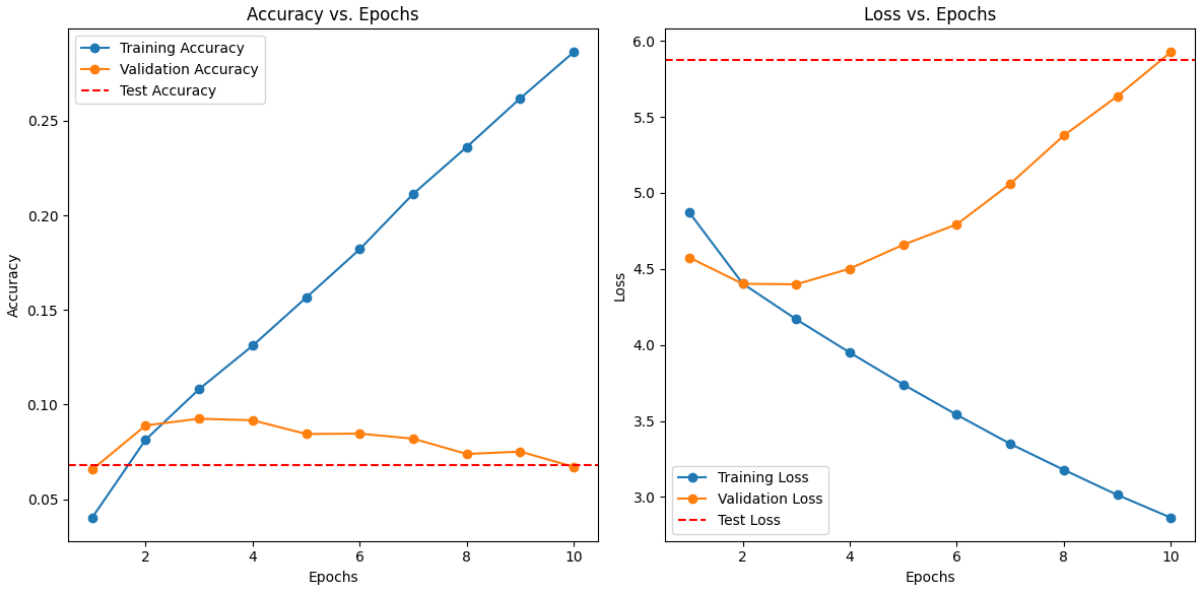
Για 5x5 μέγεθος:



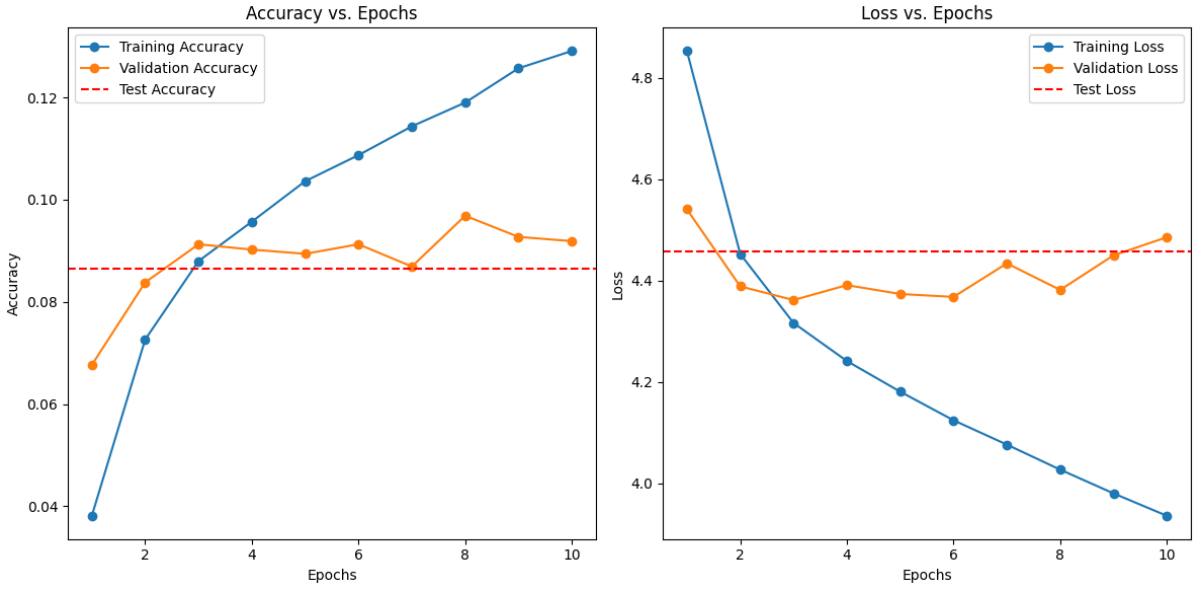
και χρόνο εκπαίδευσης 32 λεπτά, ενώ για 7x7:



και χρόνο εκπαίδευσης 28 λεπτά. Παρατηρώ ότι αυξάνοντας το μέγεθος των φίλτρων χάνεται η λεπτομέρεια στην πληροφορία που εξάγω. Αυτό αν και μειώνει τον χρόνο εκπαίδευσης λόγω μικρότερου όγκου δεδομένων, μειώνει και την ευστοχία του δικτύου στην κατηγοριοποίηση καινούργιων δειγμάτων. Επομένως κρίνεται ότι πιο αποτελεσματικό είναι το φίλτρο διάστασης 3x3. Η επόμενη παράμετρος που θα δοκιμαστεί είναι το padding να είναι ‘ίδιο’ (same padding). Με αυτή την τροποποίηση προστίθεται επιπλέον πληροφορία στα άκρα έτσι ώστε ο χάρτης χαρακτηριστικών εισόδου να έχει τις ίδιες διαστάσεις με αυτόν της εξόδου. Επίσης, σε αντίθεση με τις προηγούμενες περιπτώσεις, διατηρείται η πληροφορία στα άκρα της εικόνας επειδή το φίλτρο εμπεριέχει και τα ακραία πίξελ.

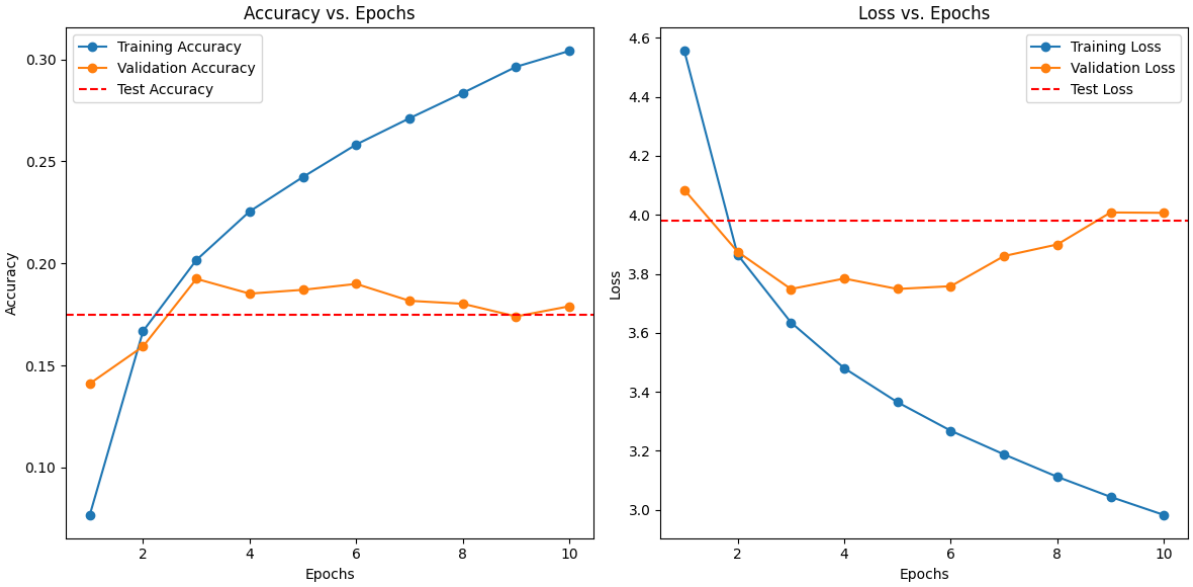


Ο χρόνος εκπαίδευσης ήταν 35 λεπτά, προφανώς μεγαλύτερος διότι πρόκειται για περισσότερους υπολογισμούς. Ωστόσο, συμπεραίνεται ότι το valid padding είναι πιο αποδοτικό, οπότε προτιμάται αυτό. Η επόμενη διαφοροποίηση είναι να αυξηθεί το stride σε 2x2. Αντί το φίλτρο να κινείται ένα πίξελ τη φορά, κινείται 2 οριζόντια και κάθετα. Αυτή η μέθοδος μειώνει τις διαστάσεις της εξόδου μειώνοντας την πληροφορία πιο επιθετικά, μειώνοντας όμως και την διαδικασία συνέλιξης ως προς τους υπολογισμούς και τον χρόνο. Μετά από 10 λεπτά τα αποτελέσματα είναι τα εξής:



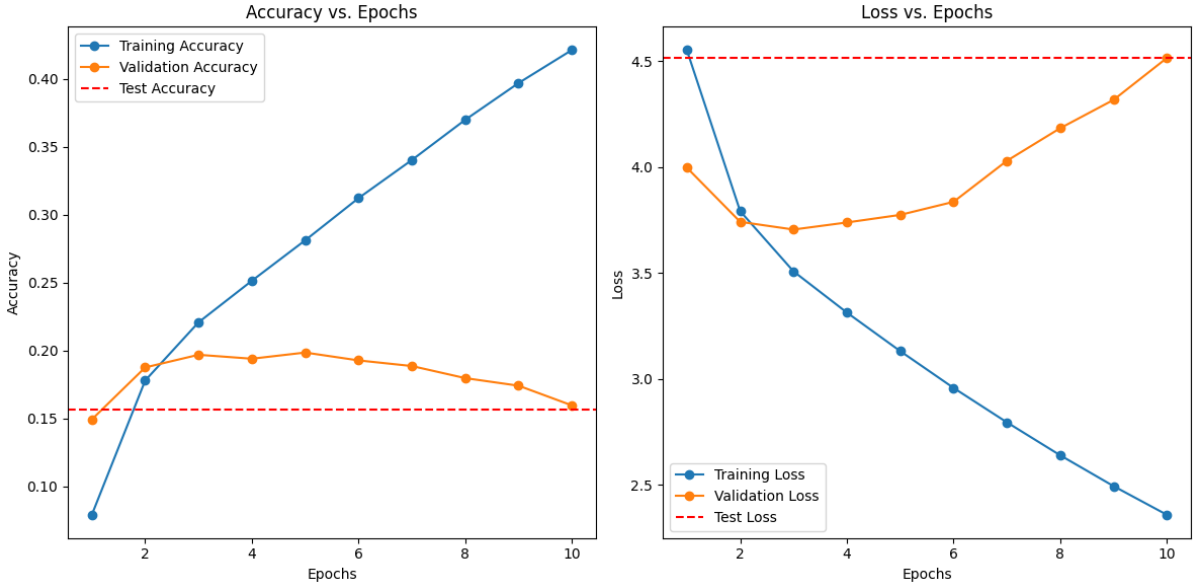
Είναι προφανές ότι υπάρχει βελτίωση της ευστοχίας. Το επόμενο βήμα είναι η προσθήκη περισσότερων συνελικτικών στρώσεων με διαφορετικούς αριθμούς φίλτρων, μεγεθών φίλτρων και stride.

- 2ο Convolutional Layer, 32 filters, 3x3 kernel size, 2x2 strides, valid padding:



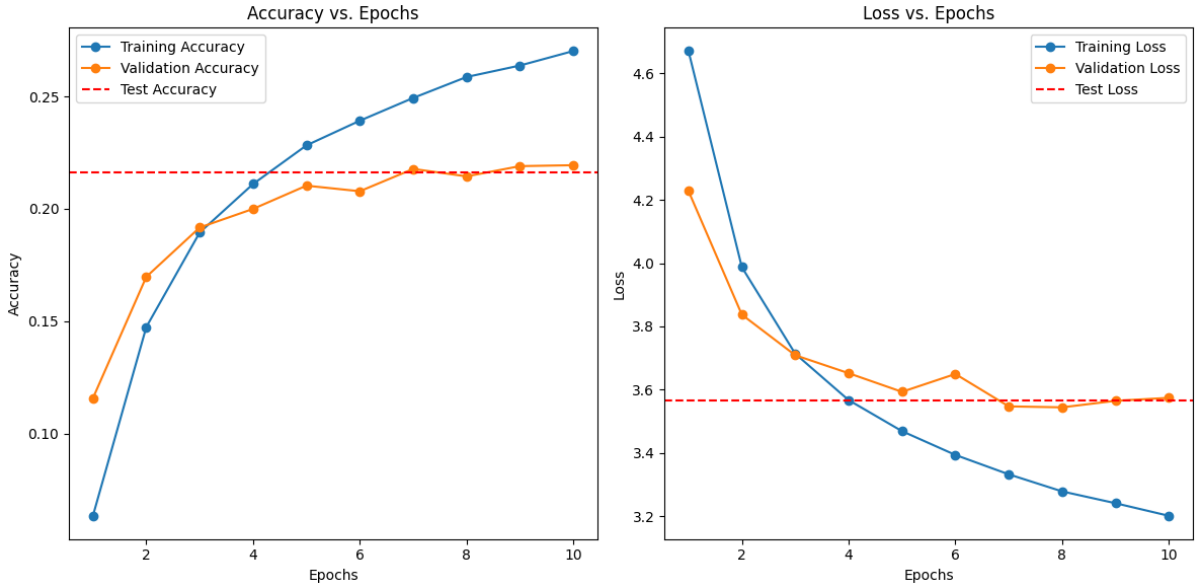
Χρόνος εκπαίδευσης: 7 λεπτά

- 2ο Convolutional Layer, 64 filters, 3x3 kernel size, 2x2 strides, valid padding:



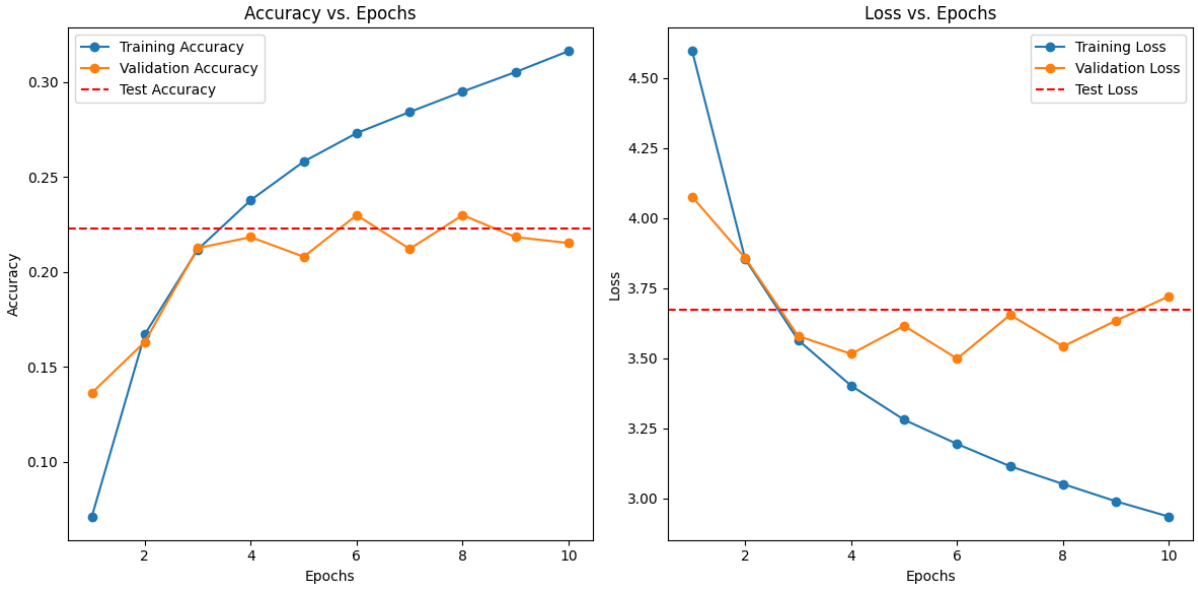
Χρόνος εκπαίδευσης: 7,5 λεπτά. Παρατηρείται ότι το δίκτυο προτιμάει 32 φίλτρα ανά layer, οπότε συνεχίζεται η διερεύνηση για το 3ο Convolutional Layer.

- 3ο Convolutional Layer, 64 filters, 3x3 kernel size, 2x2 strides, valid padding:



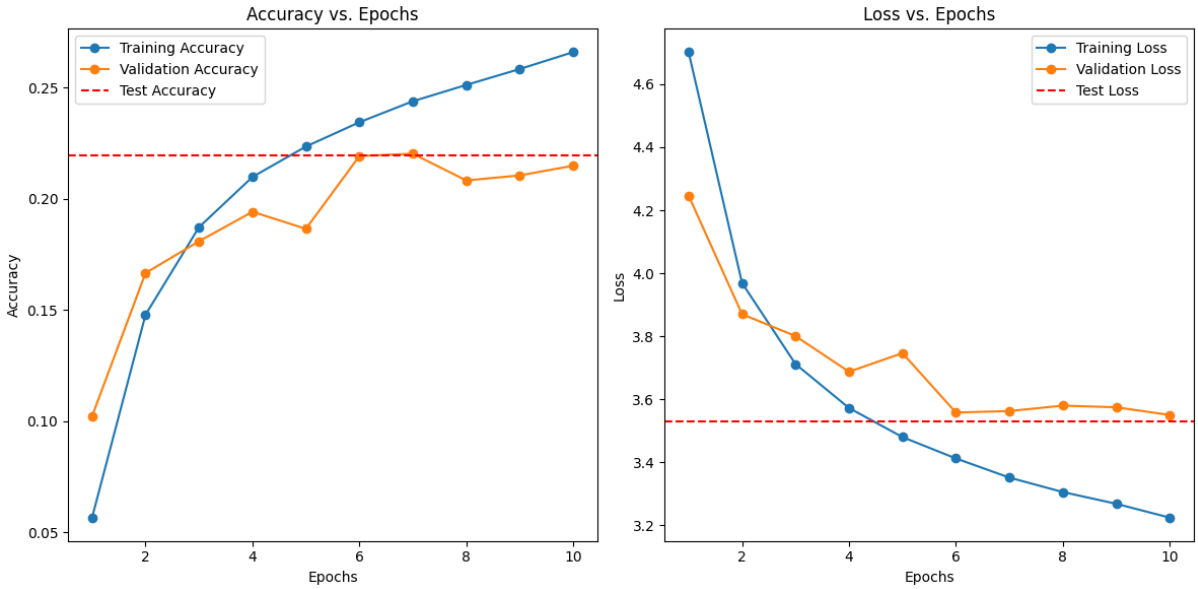
Χρόνος εκπαίδευσης: 7 λεπτά

- 3ο Convolutional Layer, 32 filters, 3x3 kernel size, 2x2 strides, valid padding:



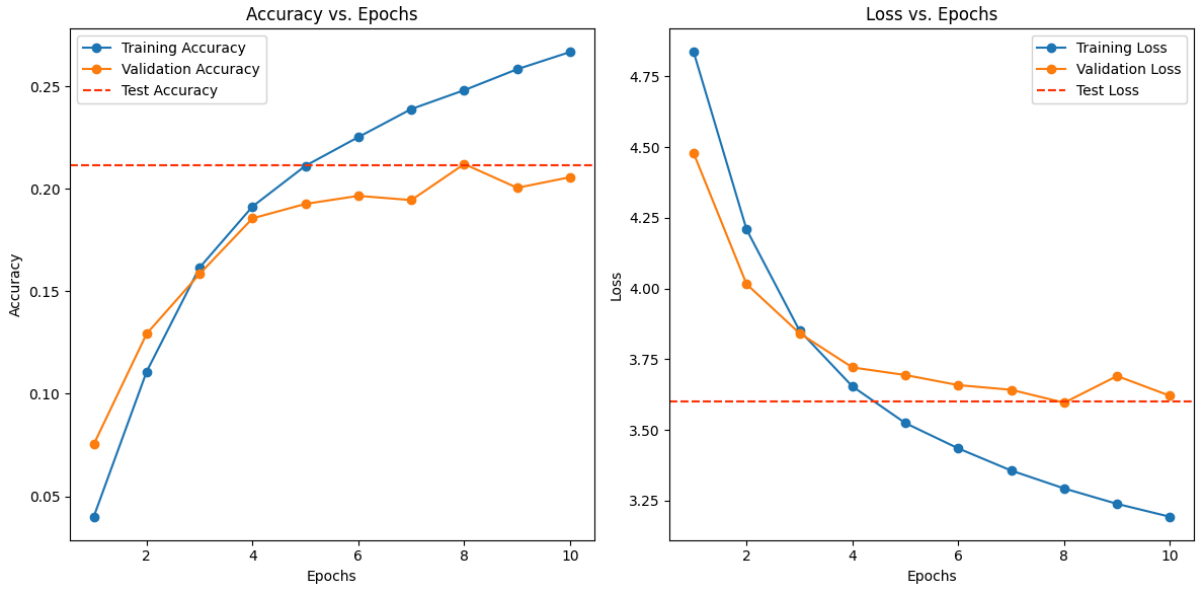
Χρόνος εκπαίδευσης: 7 λεπτά. Παρατηρώ ότι αντιδρά καλύτερα στα 64 φίλτρα πλέον το δίκτυο. Ερευνώ τα αποτελέσματα για την προσθήκη max-pooling layer πριν και μετά τη δεύτερη συνελικτική στρώση.

- Max-Pooling layer μετά τις δυο συνελικτικές και πριν την τρίτη:



Χρόνος εκπαίδευσης: 7,5 λεπτά.

- Max-Pooling layer μετά την πρώτη συνελικτική στρώση και πριν την δεύτερη:

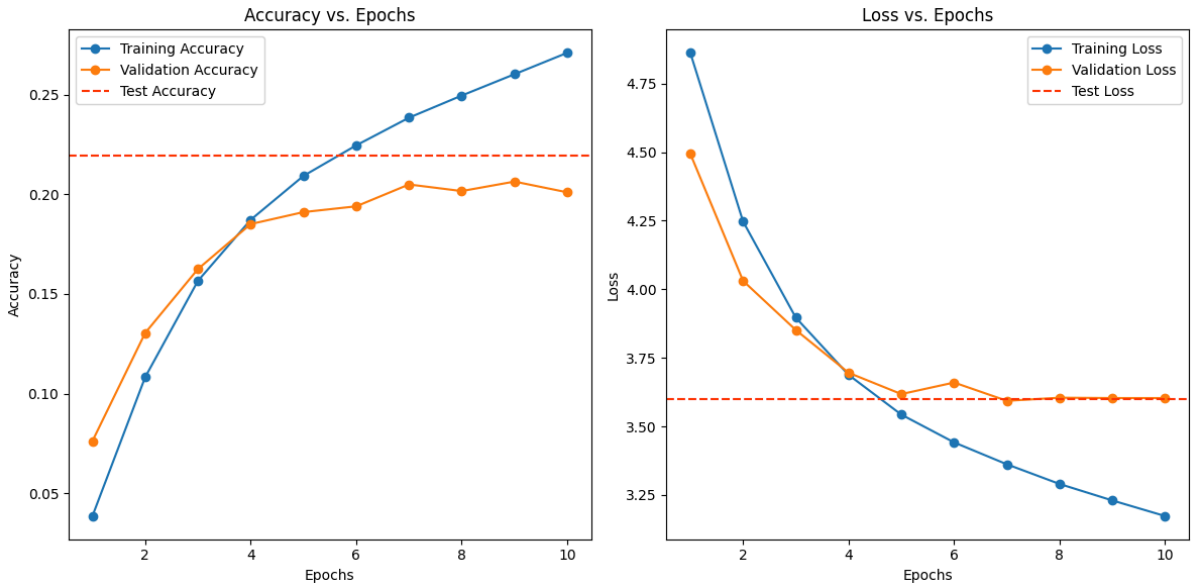


Χρόνος εκπαίδευσης: 8 λεπτά. Παρατηρώ ότι με την εισαγωγή max-pooling layer η ευστοχία μειώθηκε. Τα max-pooling layers όχι μόνο μειώνουν την υπολογιστική ισχύ, αλλά καθιστούν το δίκτυο πιο ανθεκτικό στον θόρυβο και στις μικρές παραλλαγές χαμηλώνοντας την ευκρίνεια αρκετά για να μπορεί να κατηγοριοποιεί το δίκτυο, όχι όμως τόσο ώστε να μην έχει αρκετή πληροφορία. Το ίδιο συμβαίνει με το αυξημένο stride στις συνελικτικές στρώσεις, άρα δεν χρειάζεται περαιτέρω συμπίεση της

πληροφορίας. Να σημειωθεί ότι είναι διαφορετικά τα δυο χαρακτηριστικά, ωστόσο στην συγκεκριμένη περίπτωση οι ιδιότητές τους συμπίπτουν και έχουν παρόμοια επίπτωση στην ευστοχία.

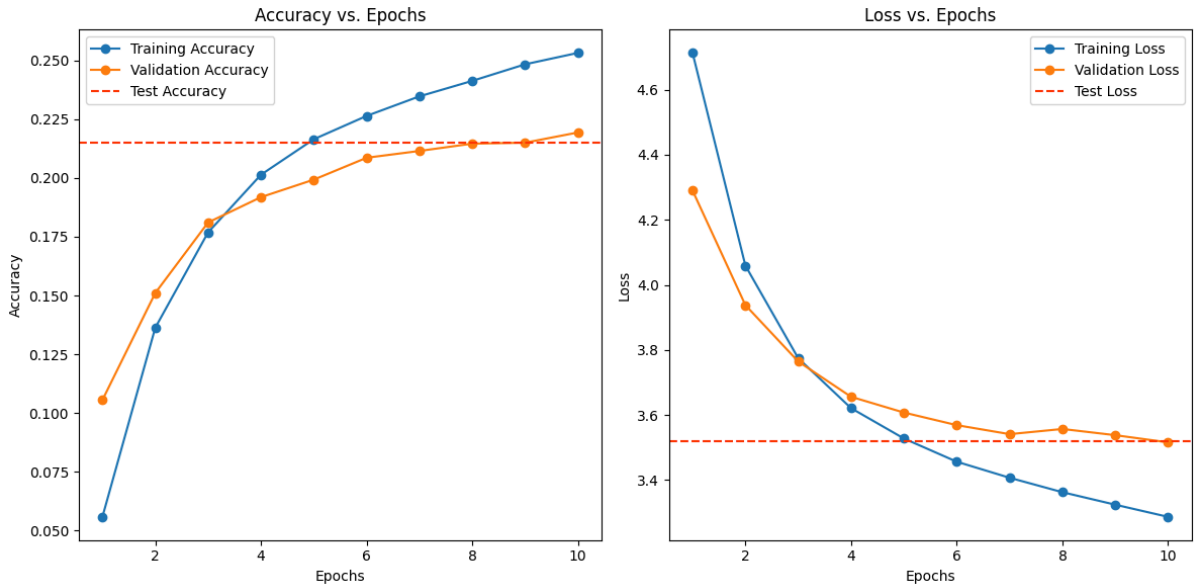
Συνεχίζω με convolutional layers μέχρι να παρατηρήσω μεγάλο overfitting ή μείωση του validation accuracy.

- 4ο Convolutional Layer, 128 filters, 3x3 kernel size, 2x2 strides, valid padding:



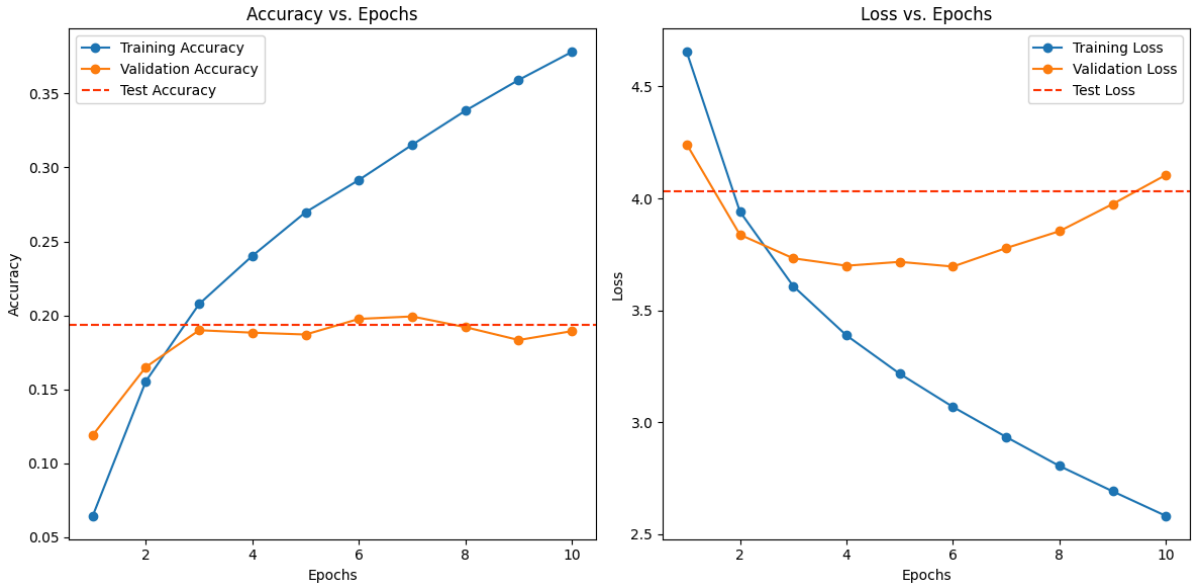
Χρόνος εκπαίδευσης: 9 λεπτά

- 4ο Convolutional Layer, 64 filters, 3x3 kernel size, 2x2 strides, valid padding:



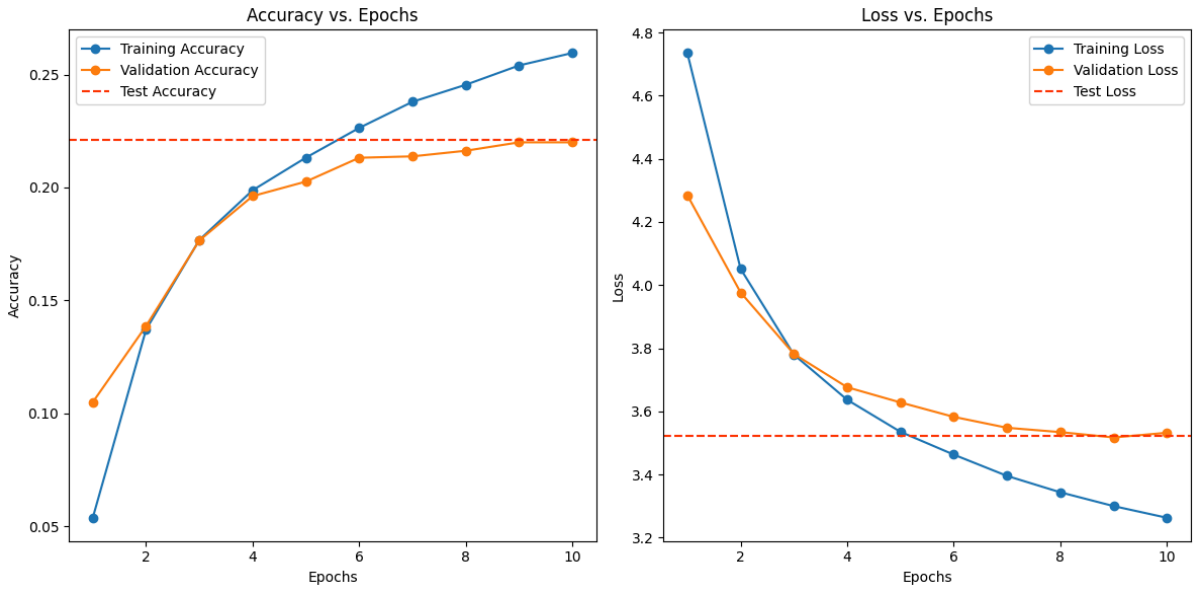
Χρόνος εκπαίδευσης: 8,5 λεπτά. Παρατηρώ ότι η ευστοχία ελέγχου σταμάτησε να ανεβαίνει, άρα καταλήγω στο συμπέρασμα ότι ο αποτελεσματικότερος συνδυασμός συνελικτικών layers είναι 3 στρώσεις με αριθμό φίλτρων 32, 32 και 64 αντίστοιχα, valid padding, $3x3$ kernel size και $2x2$ strides. Γίνονται περαιτέρω πειράματα με τα παραπάνω χαρακτηριστικά για το τελευταίο max-pooling layer.

- Χωρίς max-pooling layer



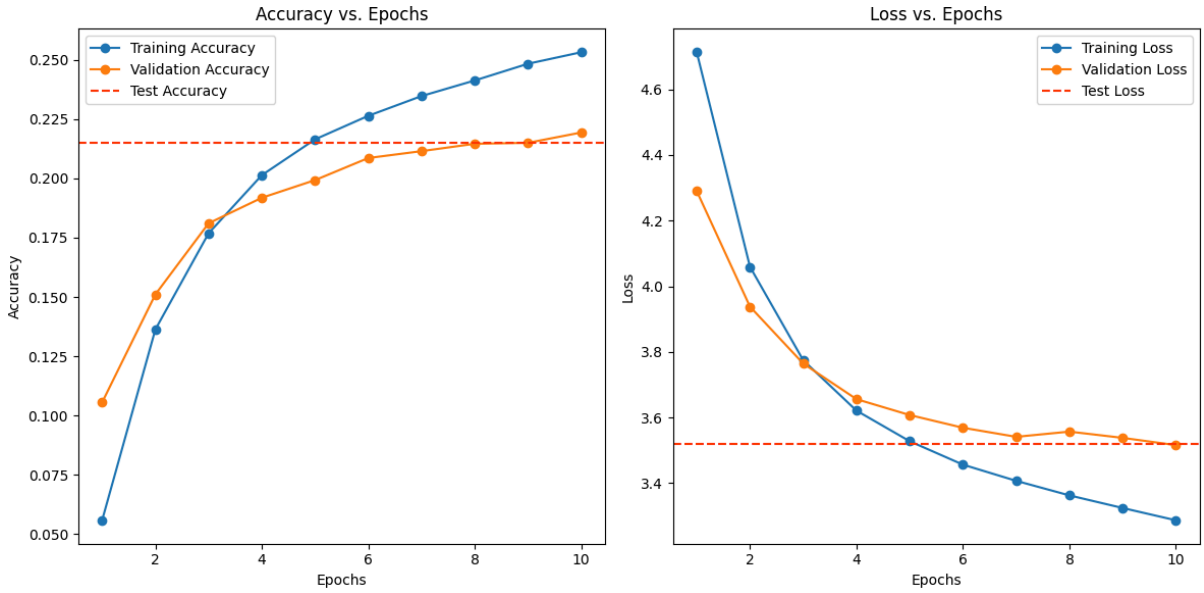
Χρόνος εκπαίδευσης: 8 λεπτά

- Με max-pooling layer με pooling size $3x3$



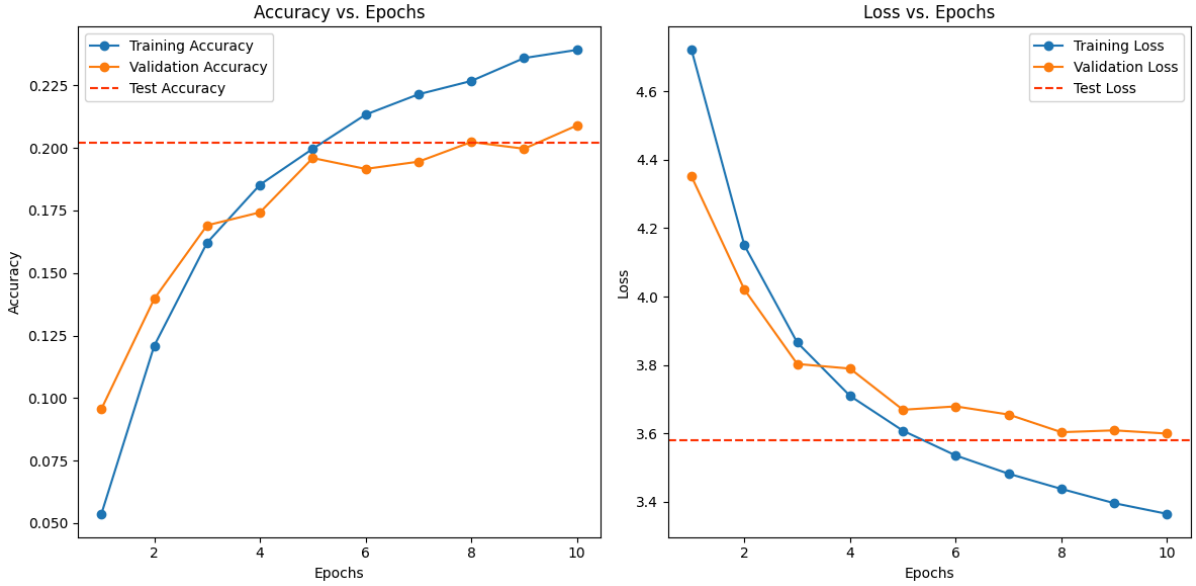
Χρόνος εκπαίδευσης: 8 λεπτά

- Με max-pooling layer με pooling size $4x4$



και χρόνο εκπαίδευσης 8 λεπτά.

- Με max-pooling layer με pooling size $5x5$



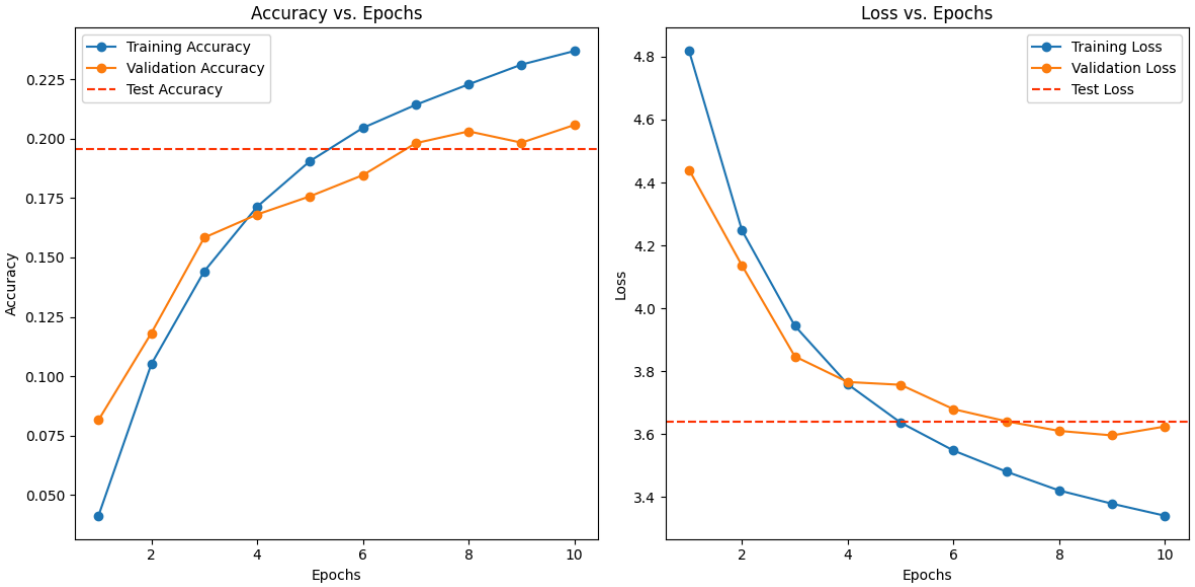
και χρόνο εκπαίδευσης 7,5 λεπτά. Παρατηρώ ότι το δίκτυο δεν βελτιώνεται με περαιτέρω τροποποιήσεις, οπότε σταματώ την διερεύνηση των συνελικτικών και max-pooling στρώσεων. Επιλέγονται 3 συνελικτικές στρώσεις με αριθμό φίλτρων 32, 32, 64 αντίστοιχα, $3x3$ μέγεθος φίλτρων, $2x2$ stride και valid padding

ακολουθούμενες από μία max-pooling στρώση μεγέθους $2x2$. Εξετάζω πλέον τις πλήρως συνδεδεμένες στρώσεις.

- **Πλήρως Συνδεδεμένες Στρώσεις**

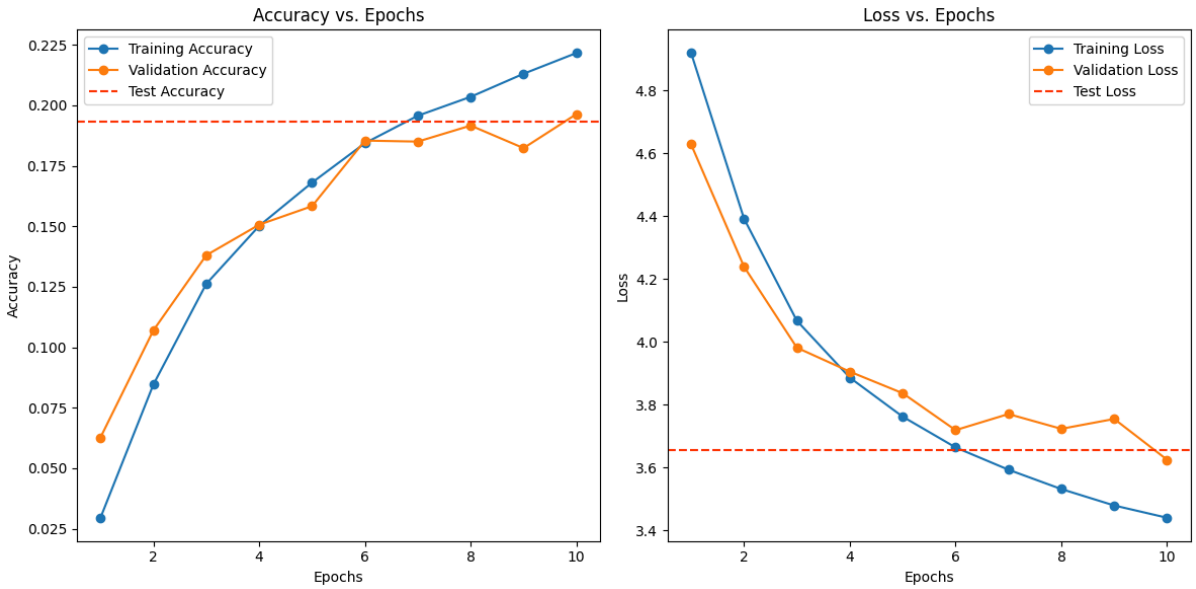
Ξεκινώντας από μία στρώση με 128 νευρώνες γίνονται διάφορες δοκιμές.

- $2^{η}$ Πλήρως Συνδεδεμένη Στρώση με 256 νευρώνες



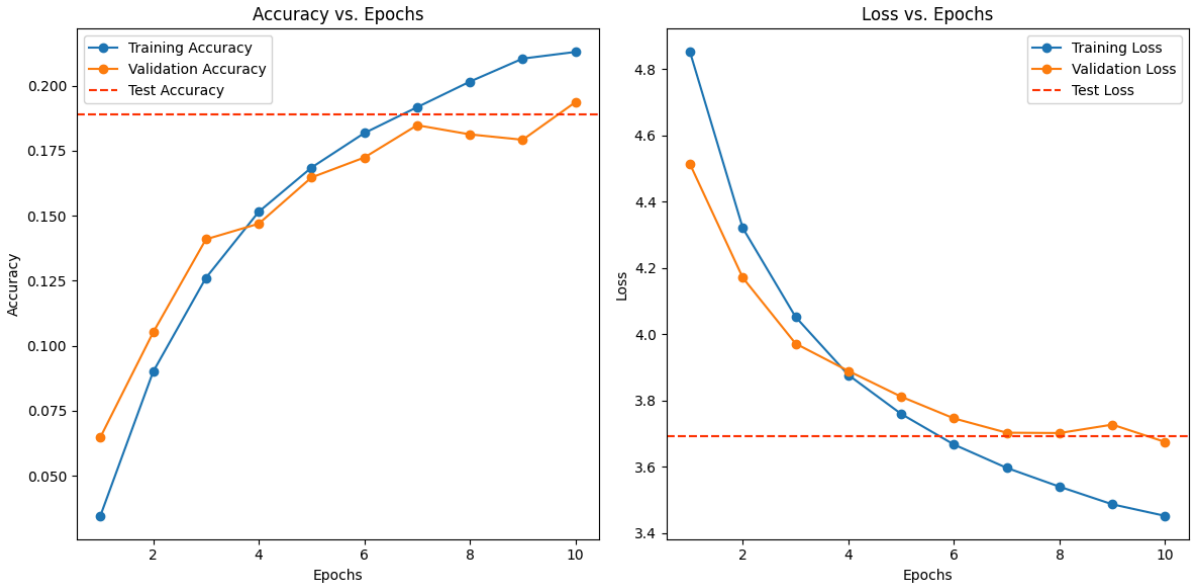
Χρόνος Εκπαίδευσης: 8,5 λεπτά

- $2^{η}$ Πλήρως Συνδεδεμένη Στρώση με 128 νευρώνες



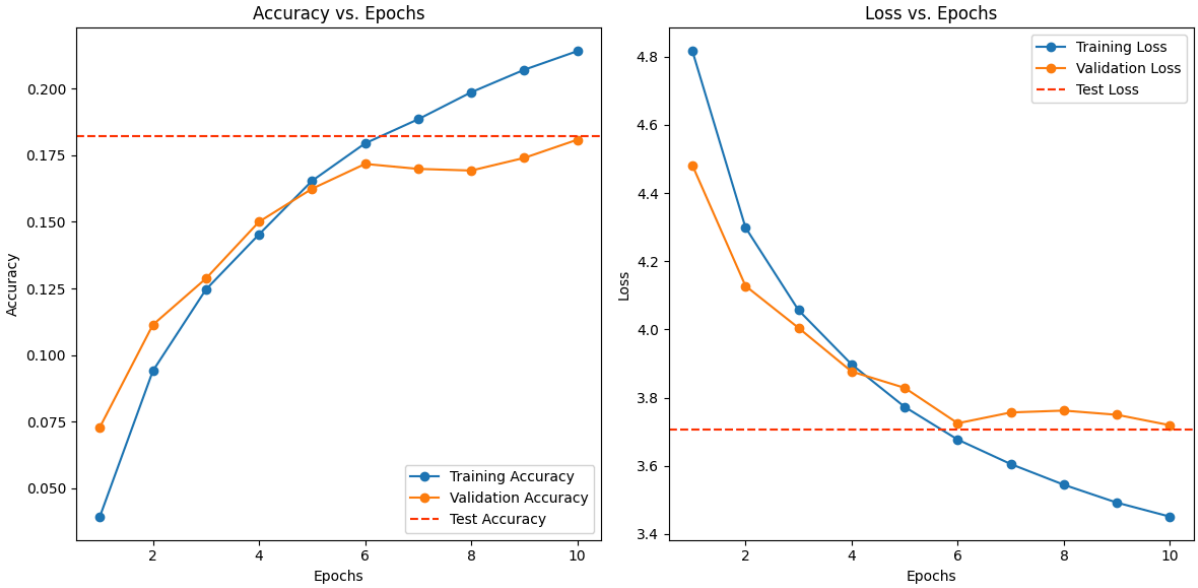
Χρόνος εκπαίδευσης: 9 λεπτά

- $2^{η}$ και $3^{η}$ Πλήρως Συνδεδεμένη Στρώση με 128 νευρώνες



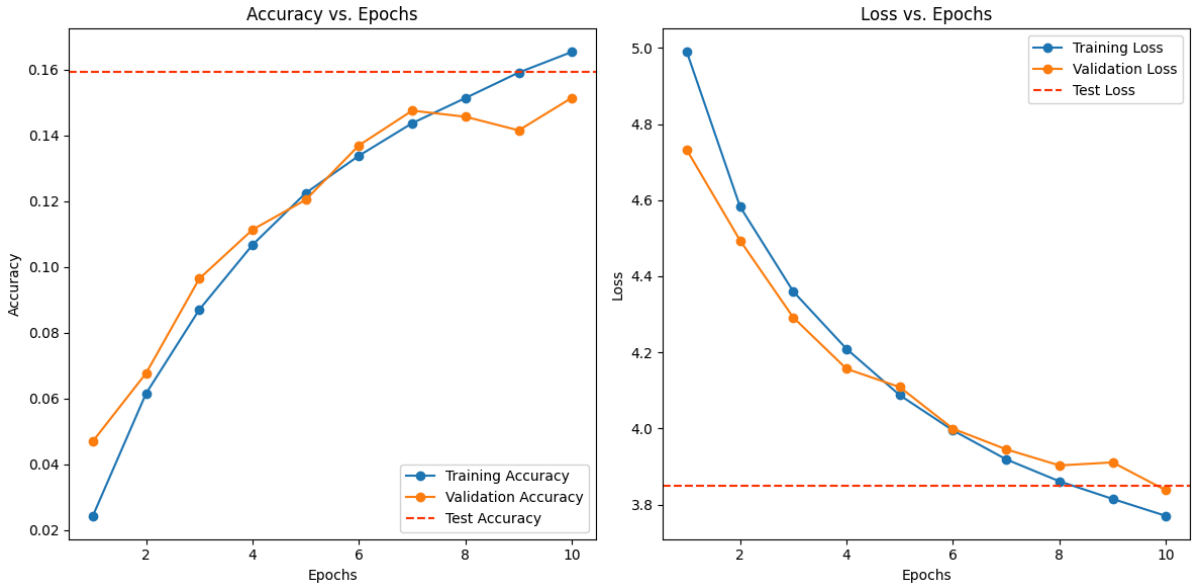
Χρόνος Εκπαίδευσης: 9 λεπτά

- $2^{η}$ και $3^{η}$ Πλήρως Συνδεδεμένη Στρώση με 128 και 256 νευρώνες



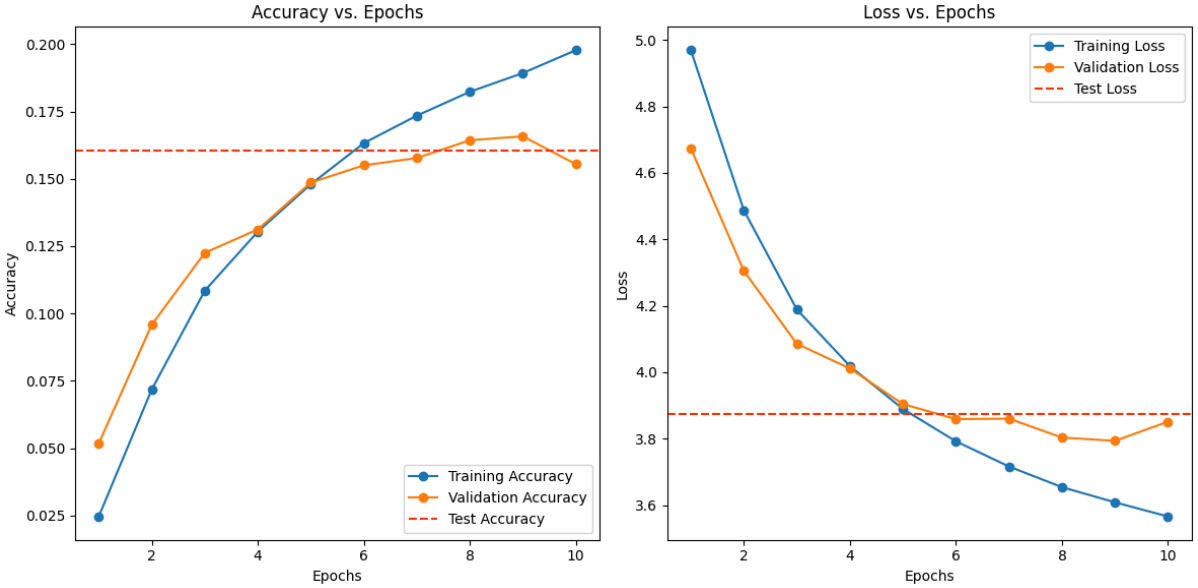
Χρόνος Εκπαίδευσης: 9 λεπτά

- 1 ^{η}, 2^{η} και 3^{η} Πλήρως Συνδεδεμένη Στρώση με 64, 128 και 256 νευρώνες



Χρόνος Εκπαίδευσης: 9 λεπτά

- 2^{η} και 3^{η} Πλήρως Συνδεδεμένη Στρώση με 128 και 64 νευρώνες



Χρόνος Εκπαίδευσης: 9,5 λεπτά

Παρατηρώ ότι καμία παραλλαγή των dense layers δεν αυξάνει την ευστοχία του μοντέλου, αντιθέτως αυτή μειώνεται. Τα συνελικτικά στρώματα μειώνουν υπερβολικά τις χωρικές πληροφορίες στους χάρτες χαρακτηριστικών, αφήνοντας ελάχιστες πληροφορίες για τα πυκνά στρώματα προς επεξεργασία. Αυτό συμβαίνει

συνήθως όταν υπάρχει σημαντική μείωση μεγέθους της πληροφορίας. Πάρα πολλά επίπεδα συνέλιξης και max-pooling μειώνουν προοδευτικά τις χωρικές διαστάσεις των δεδομένων. Πράγματι, κοιτώντας τον παρακάτω πίνακα, τα 12.288 χαρακτηριστικά εισόδου μειώνονται σε μόλις 576 λόγω της μείωσης της πληροφορίας εξαιτίας των strides των συνελικτικών στρώσεων.

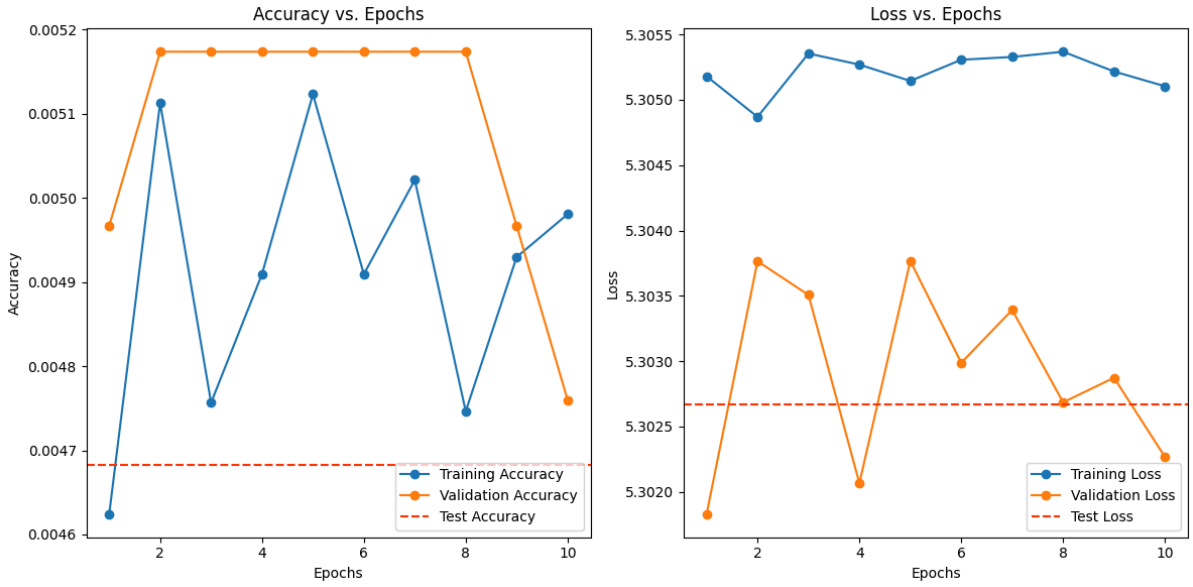
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 31, 31, 32)	896
conv2d_1 (Conv2D)	(None, 15, 15, 32)	9,248
conv2d_2 (Conv2D)	(None, 7, 7, 64)	18,496
max_pooling2d (MaxPooling2D)	(None, 3, 3, 64)	0
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 128)	73,856
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 128)	8,320
dense_3 (Dense)	(None, 200)	25,800

Επομένως, αυξημένο πλήθος πλήρως συνδεδεμένων στρώσεων δεν θα αυξήσει την ευστοχία του δικτύου. Άρα, η δομή του νευρωνικού είναι η εξής:

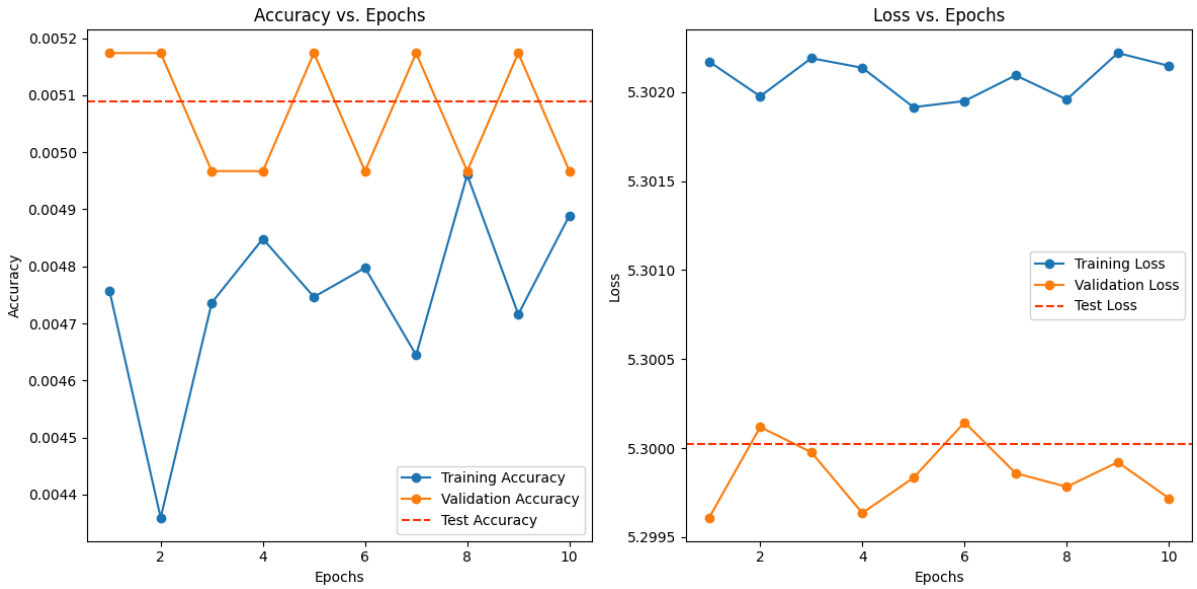
- Convolutional Layer, 32 filters, $3x3$ filter size, $2x2$ stride, valid padding, ReLU activation function
- Convolutional Layer, 32 filters, $3x3$ filter size, $2x2$ stride, valid padding, ReLU activation function
- Convolutional Layer 64 filters, $3x3$ filter size, $2x2$ stride, valid padding, ReLU activation function
- Max Pooling Layer, $2x2$ pool size, $1x1$ stride size, valid padding,
- Dense Layer, 128 neurons, ReLU activation function
- Output Layer, 200 neurons, Softmax activation function

- **Ρυθμός Μάθησης**

Επόμενη παράμετρος που θα εξεταστεί είναι ο ρυθμός μάθησης. Στο παραπάνω δίκτυο, αλλάζοντας τον ρυθμό μάθησης από 0.001 σε 0.01 παρατηρείται ταλάντωση:

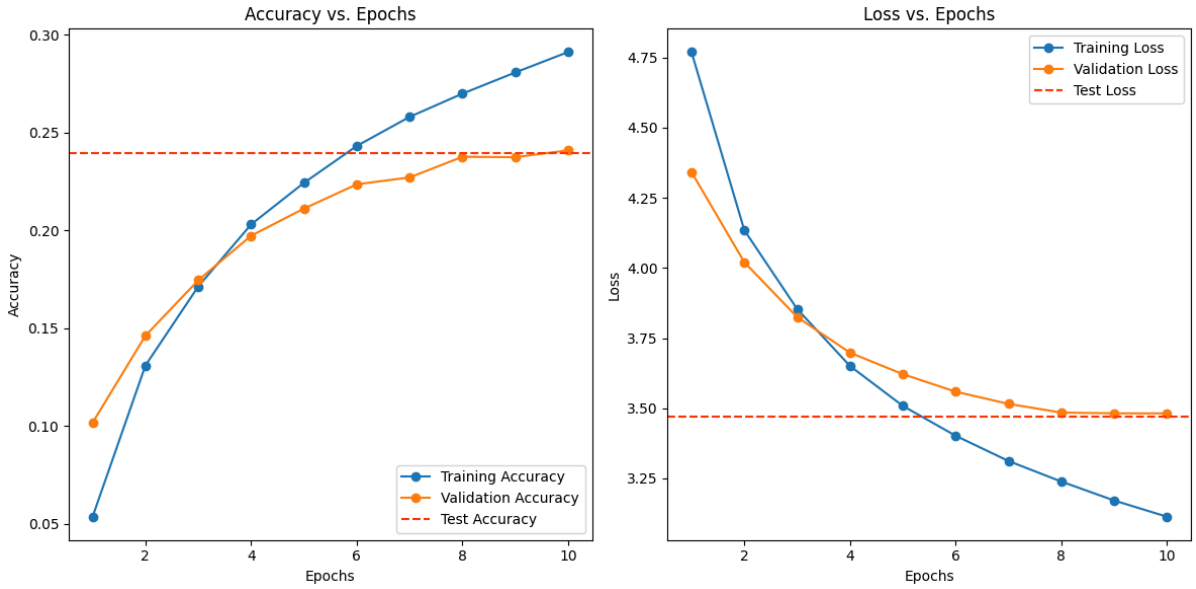


Ενώ για ρυθμό μάθησης 0.005 αντίστοιχα:

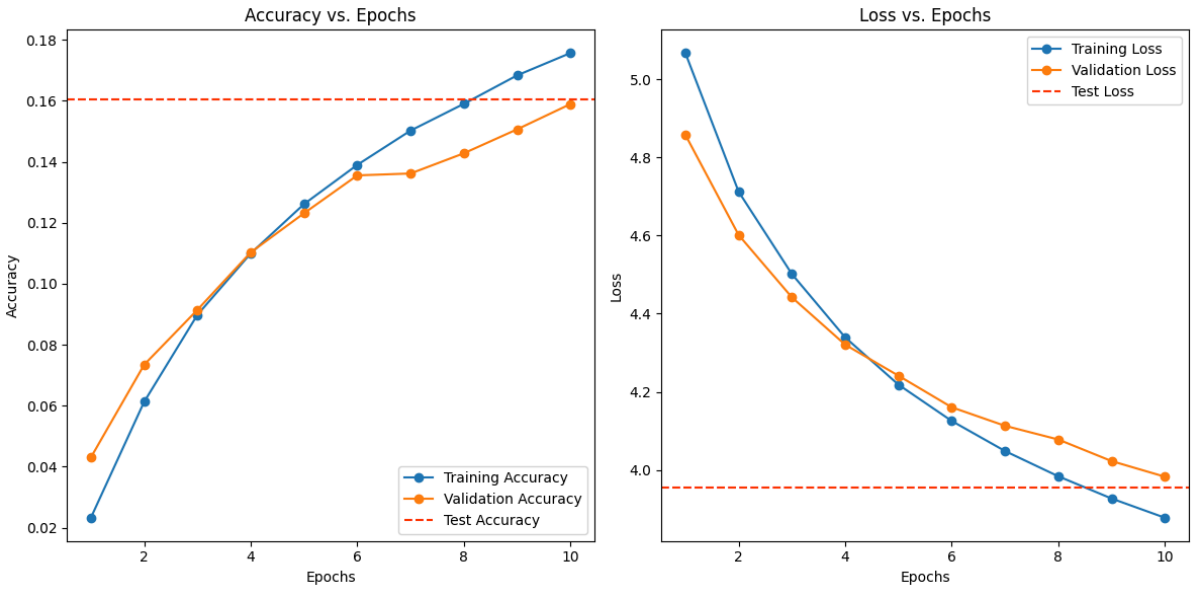


Δοκιμάζω για μικρότερους ρυθμούς μάθησης.

- $\lambda = 0.0005$



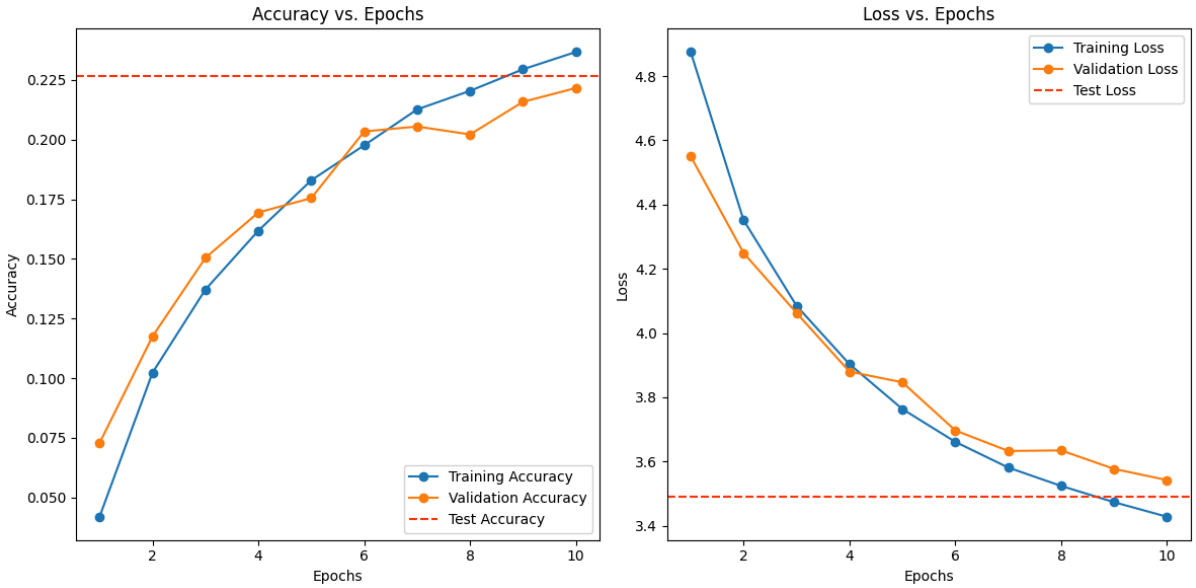
- $\lambda = 0.0001$



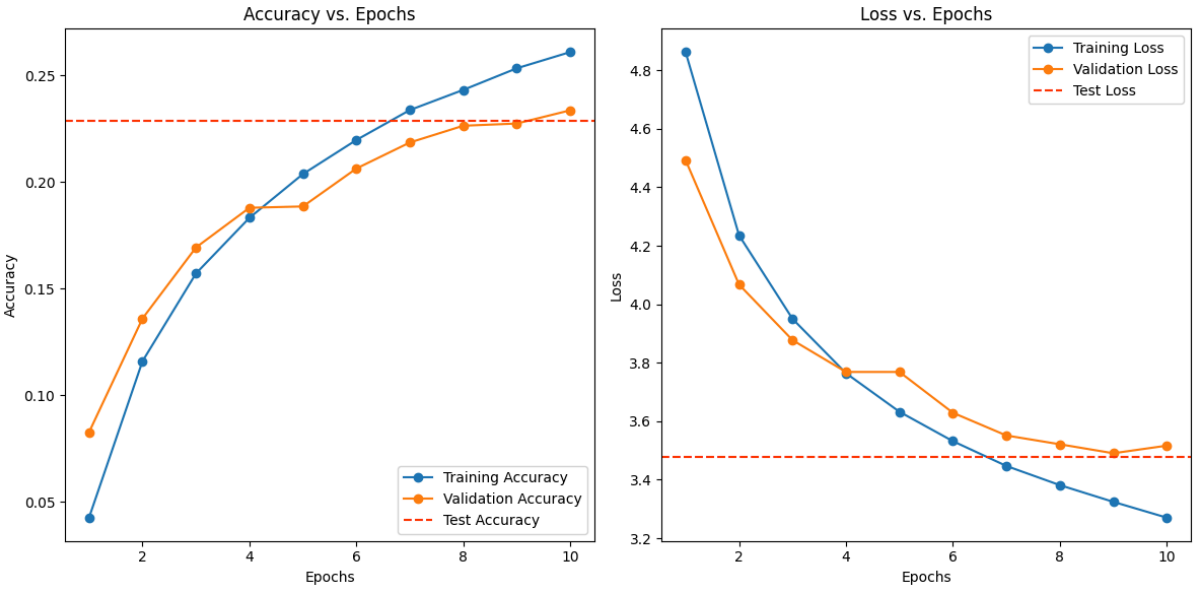
Λόγω της κλίμακας δεν είναι εμφανές, αλλά η ευστοχία ανέβηκε στο 26,6% για $\lambda = 0.0005$.

- **Κανονικοποίηση**

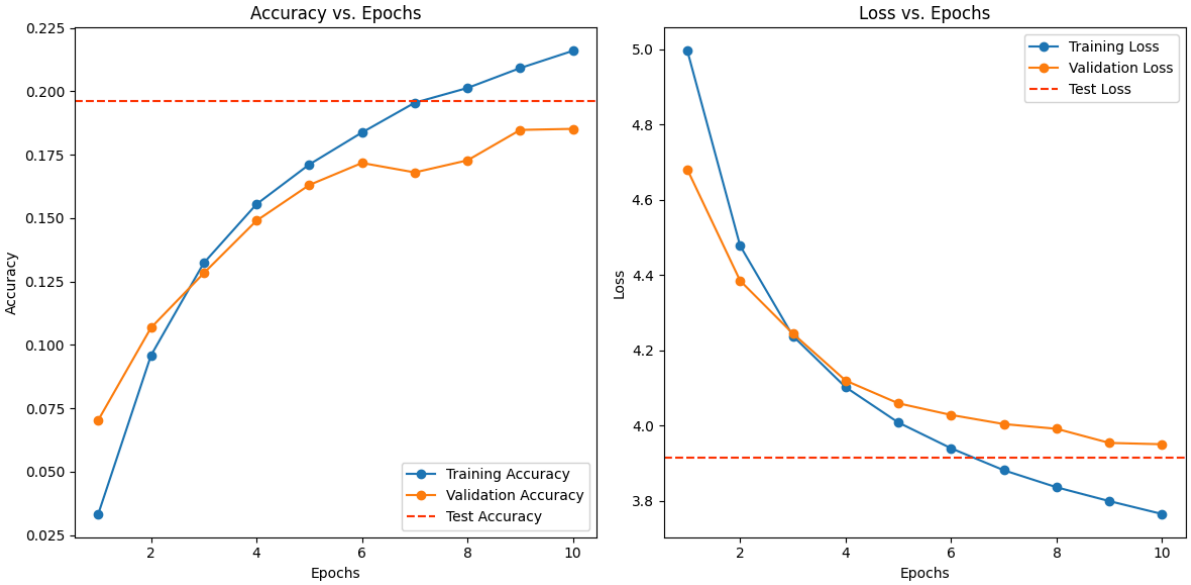
Επόμενη παράμετρος που θα εξεταστεί είναι η τεχνική κανονικοποίησης Dropout. Κατά τη διάρκεια της εκπαίδευσης, τυχαία “απενεργοποιείται” ένα ποσοστό από τους νευρώνες με μηδενισμό της τιμής τους για να εξασφαλιστεί ότι το δίκτυο δεν εξαρτάται υπερβολικά από συγκεκριμένους νευρώνες. Δοκιμάζω 0.2 μετά από κάθε συνελικτική στρώση.



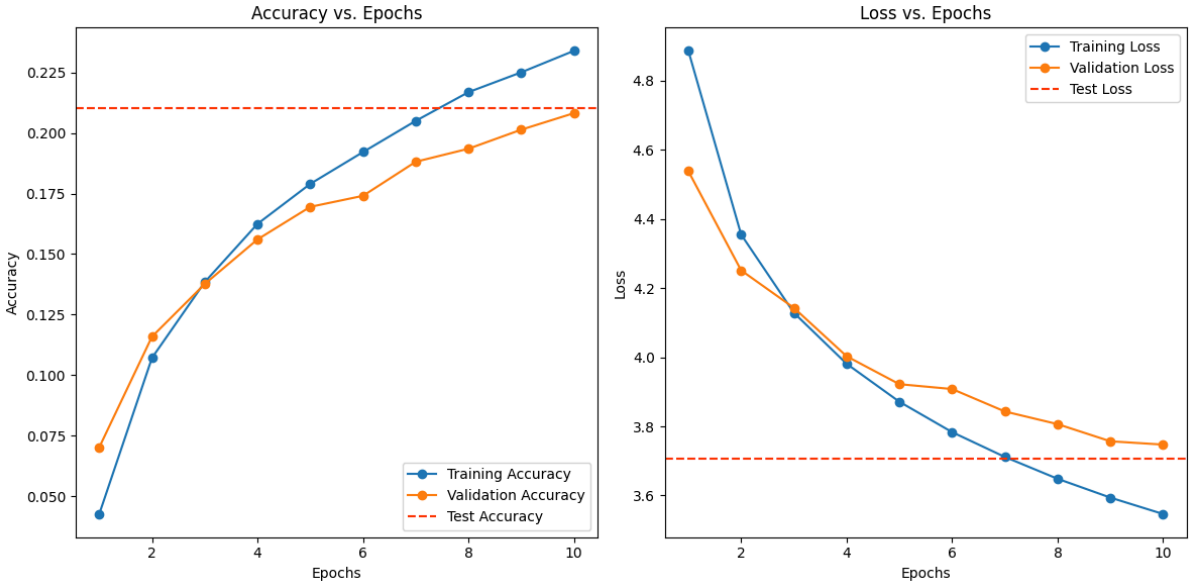
Με χρόνο εκπαίδευσης 8,5 λεπτά. Είναι φανερό από τα διαγράμματα ότι έχει μειωθεί το φαινόμενο overfitting, καθώς φαίνεται να μειώνεται η απόσταση των καμπυλών εκπαίδευσης και επαλήθευσης. Για 0.05 μετά από κάθε συνελικτική στρώση:



με χρόνο εκπαίδευσης 8,5 λεπτά. Το overfitting όντως μειώνεται, ρίχνοντας όμως την απόδοση του νευρωνικού. Δοκιμάζω L2 κανονικοποίηση με αποσύνθεση βάρους 0.001 σε κάθε συνελικτική στρώση.



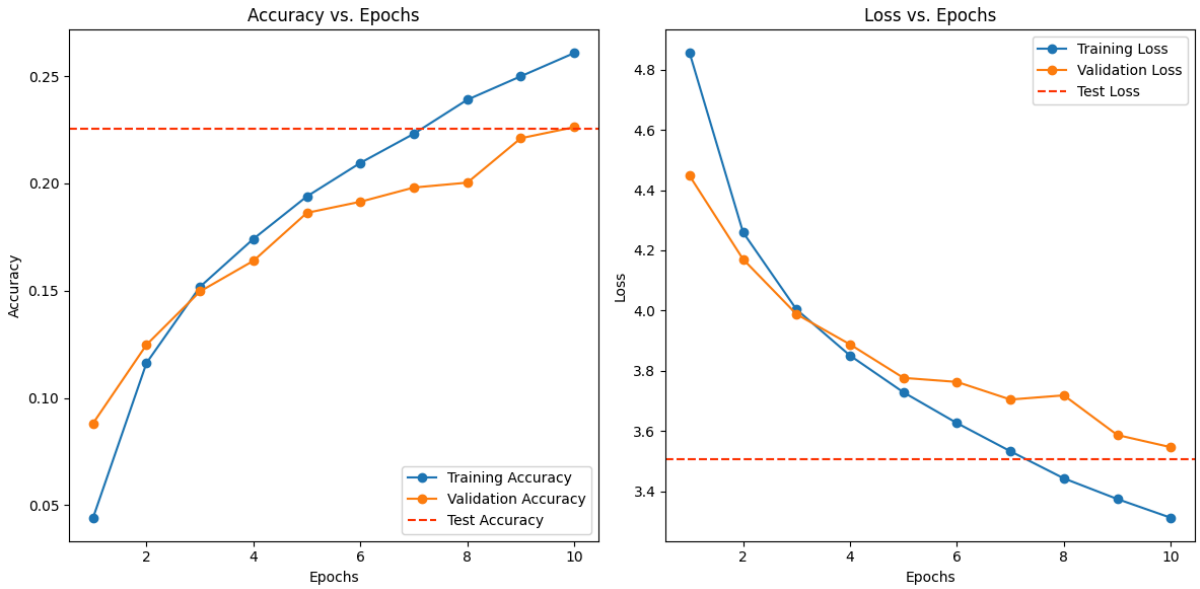
Με χρόνο εκπαίδευσης 7,5 λεπτά. Η ευστοχία μειώθηκε, θα δοκιμαστεί L2 κανονικοποίηση με αποσύνθεση βάρους 0.001 σε κάθε στρώση.



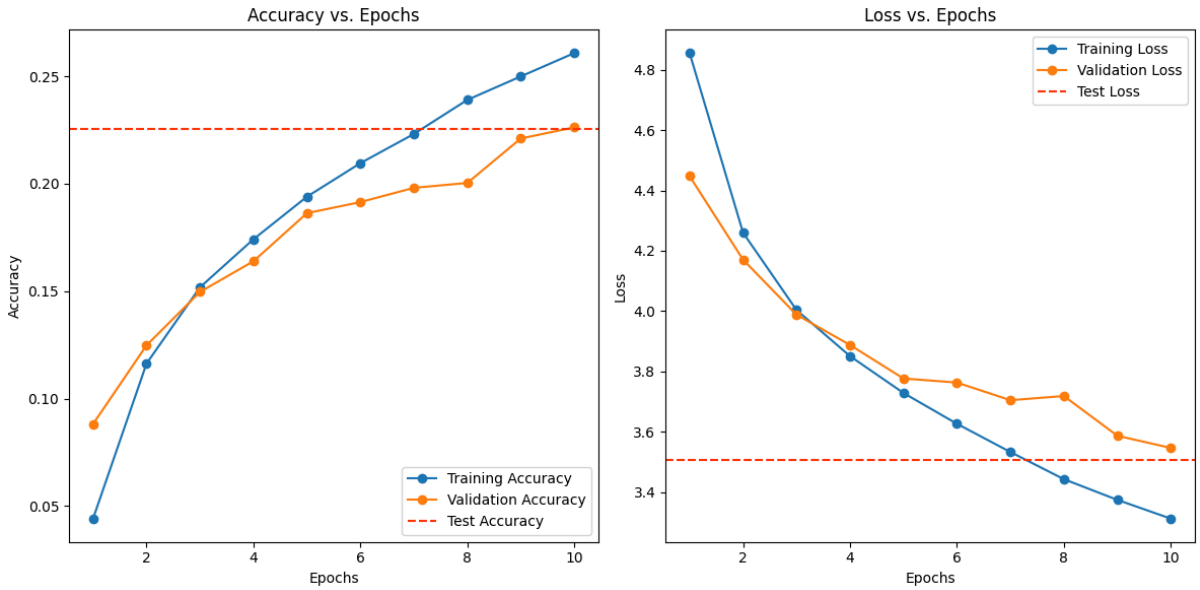
Η ευστοχία χαμηλώθηκε παραπάνω. Επομένως η κανονικοποίηση L2 δεν είναι αποτελεσματική για το δίκτυο.

- **Μέγεθος Δεσμίδων**

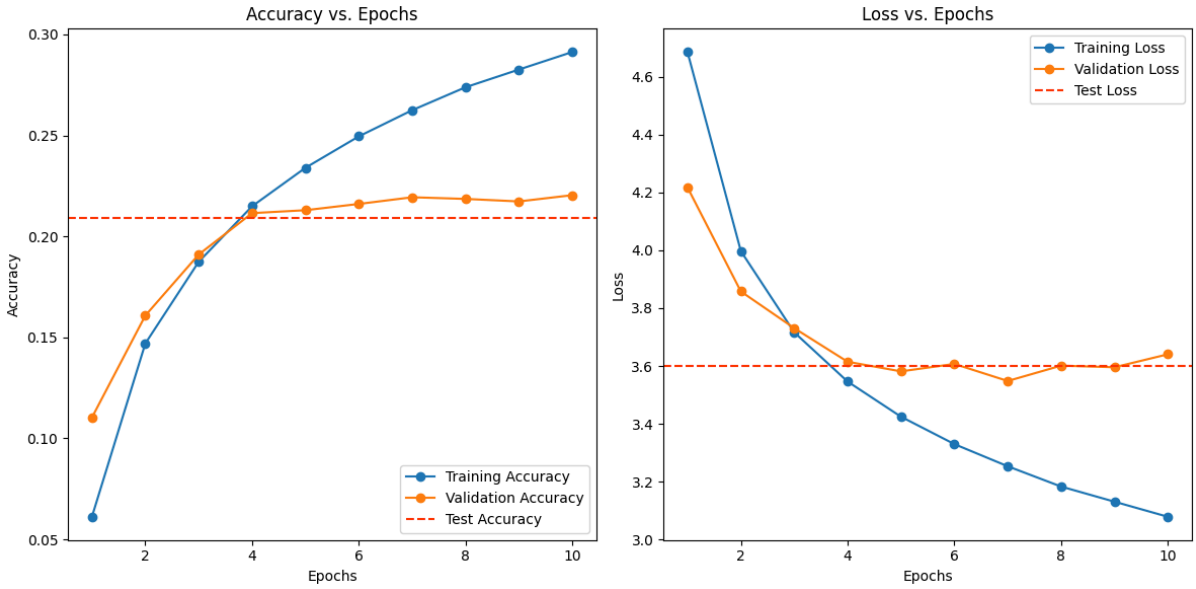
Η επόμενη διαφοροποίηση είναι το batch size. Έως τώρα χρησιμοποιούνταν batch size 32, η πρώτη δοκιμή γίνεται με 64.



Με χρόνο εκπαίδευσης 4,5 λεπτά. Η ευστοχία μειώθηκε ελάχιστα. Δοκιμάζω για την τιμή 128.



Με χρόνο εκπαίδευσης 3 λεπτά. Ο χρόνος εκπαίδευσης εμφανώς μειώνεται, όπως και η ευστοχία, γεγονός προς αποφυγή. Δοκιμάζεται batch size 16.

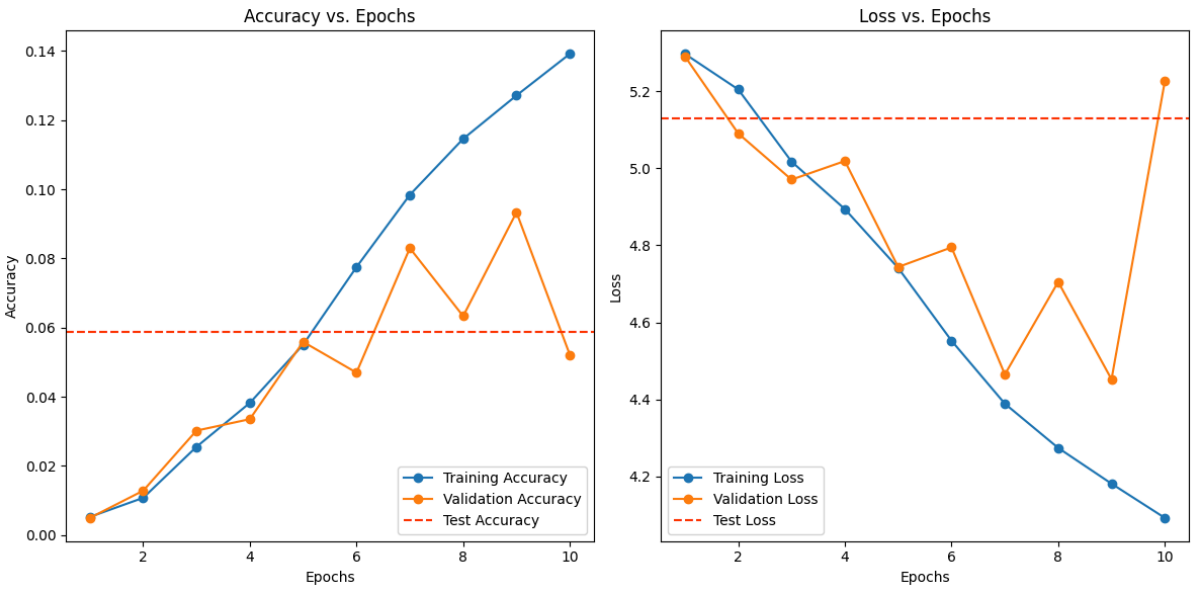


Με χρόνο εκπαίδευσης 12 λεπτά. Η ευστοχία παραμένει ελάχιστα μειωμένη, με αυξημένο χρόνο εκπαίδευσης. Για μέγεθος 8 η ευστοχία μειώνεται και ο χρόνος εκπαίδευσης αυξάνεται στα 23 λεπτά. Επομένως, επικρατεί το αρχικό batch size 32.

- **Βελτιστοποιητής**

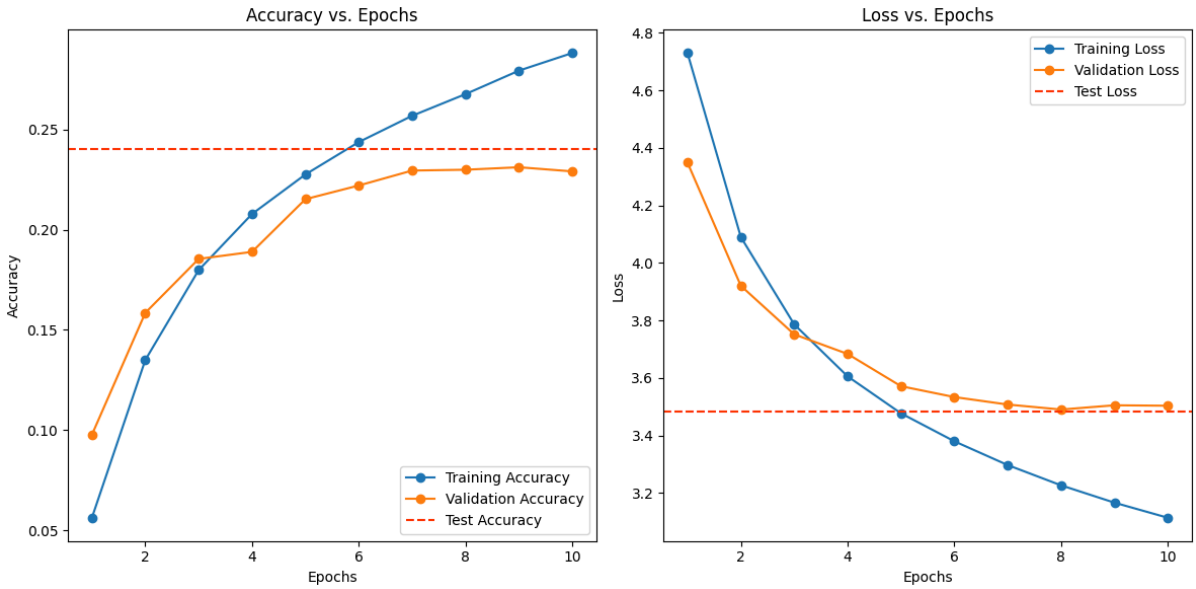
Η επόμενη παράμετρος είναι ο βελτιστοποιητής (optimizer). Στις παραπάνω δοκιμές χρησιμοποιήθηκε ο Adam, διότι είναι ο πιο προσαρμοστικός. Θα πραγματοποιηθούν δοκιμές με τους παρακάτω βελτιστοποιητές:

- SGD με ρυθμό μάθησης 0.01



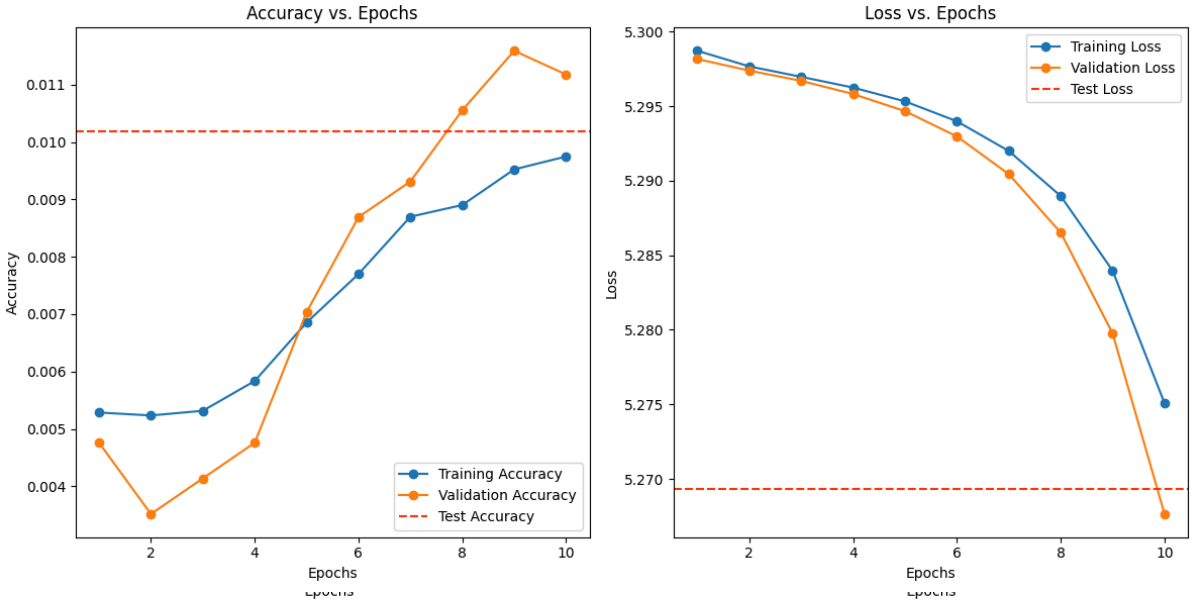
Με χρόνο εκπαίδευσης 6 λεπτά.

- SGD με ρυθμό μάθησης 0.001



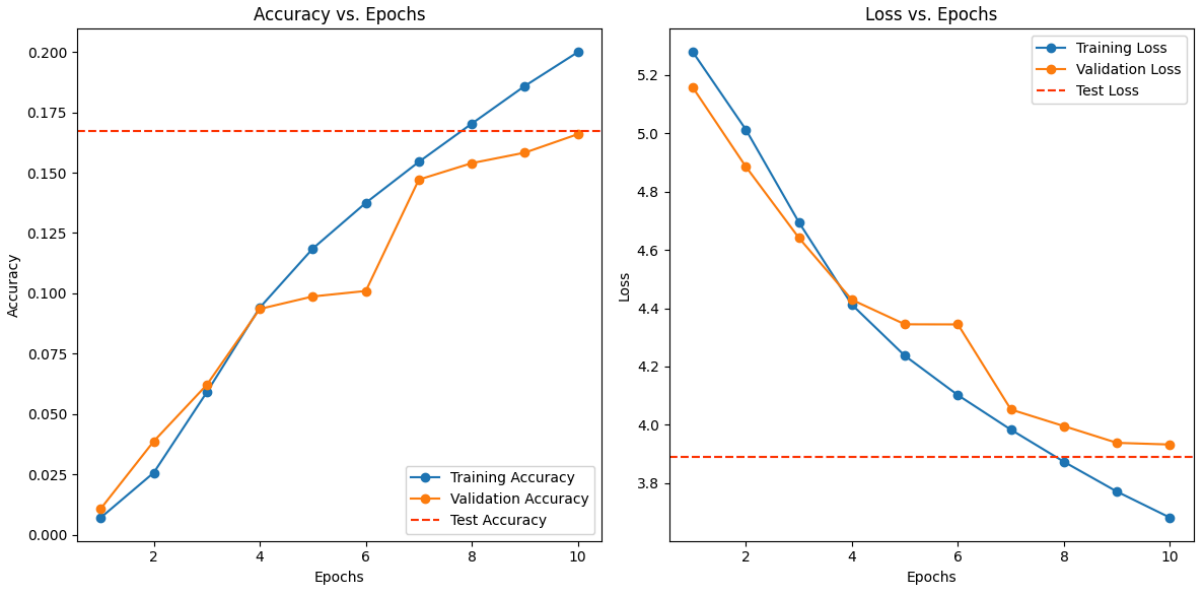
Με χρόνο εκπαίδευσης 5,5 λεπτά.

- SGD με ρυθμό μάθησης 0.01 και ορμή 0.9



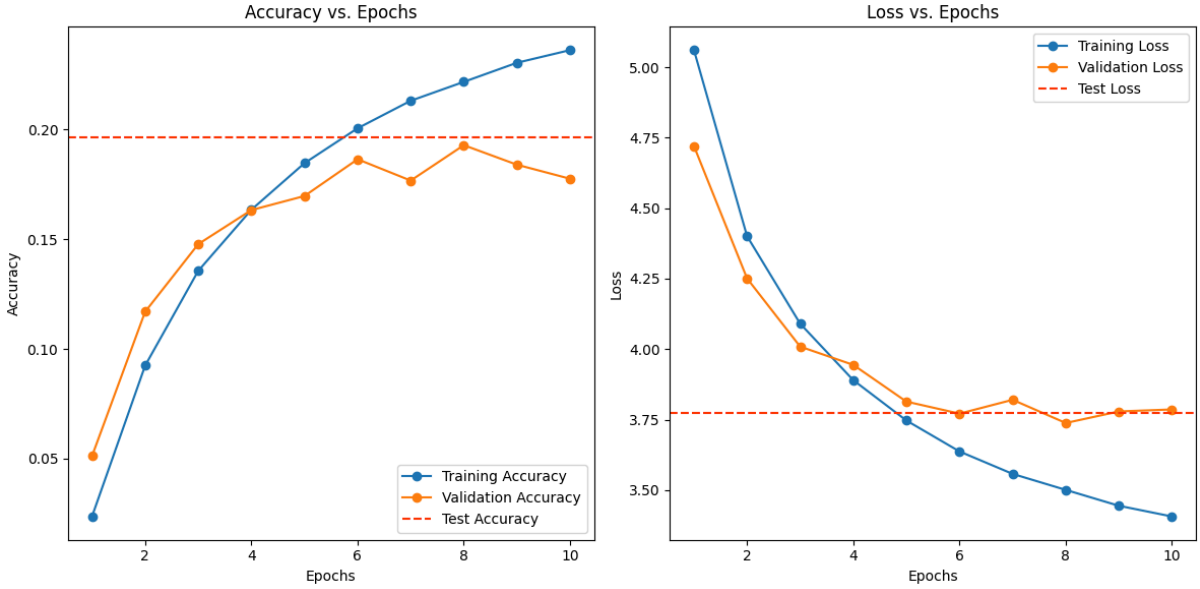
Με χρόνο εκπαίδευσης 5,5 λεπτά

- SGD με ρυθμό μάθησης 0.01 και ορμή 0.5



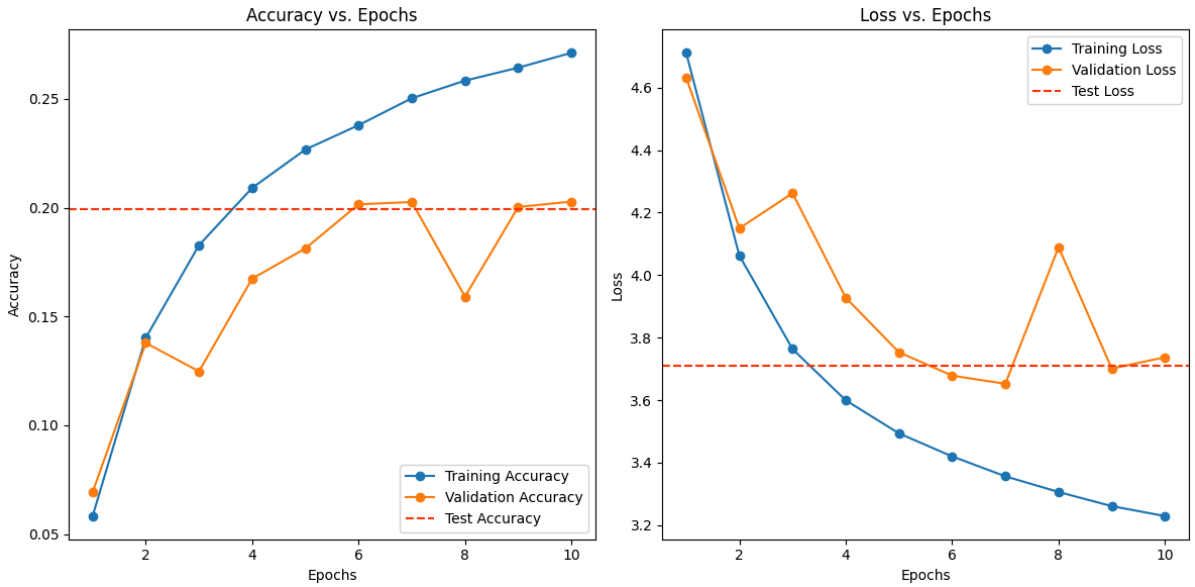
- Χρόνος εκπαίδευσης: 5,5 λεπτά

- SGD με ρυθμό μάθησης 0.01, ορμή 0.9 και αποσύνθεση βαρών 10^{-6}



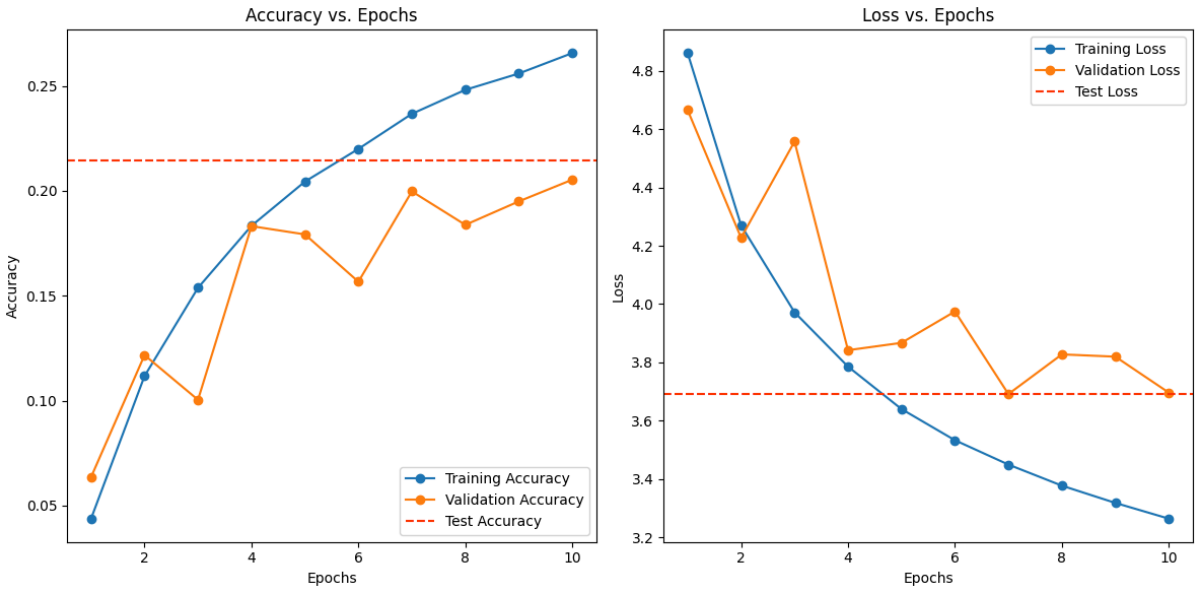
- Χρόνος εκπαίδευσης: 6 λεπτά

- RMSProp με ρυθμό μάθησης 0.001



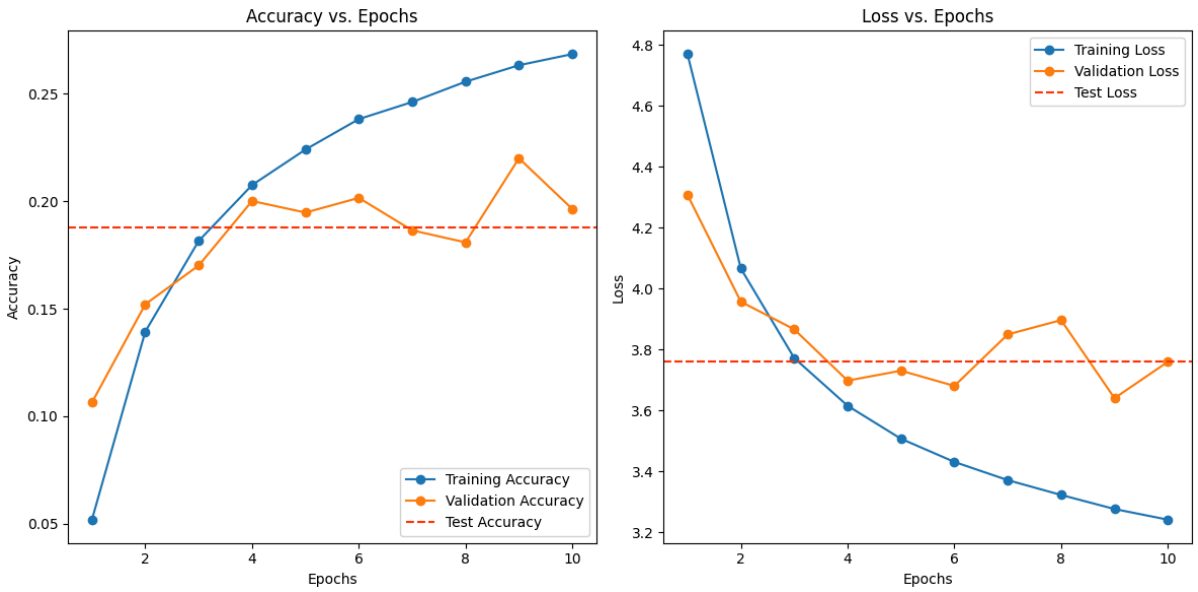
Χρόνος Εκπαίδευσης: 6 λεπτά

- RMSProp με ρυθμό μάθησης 0.0005

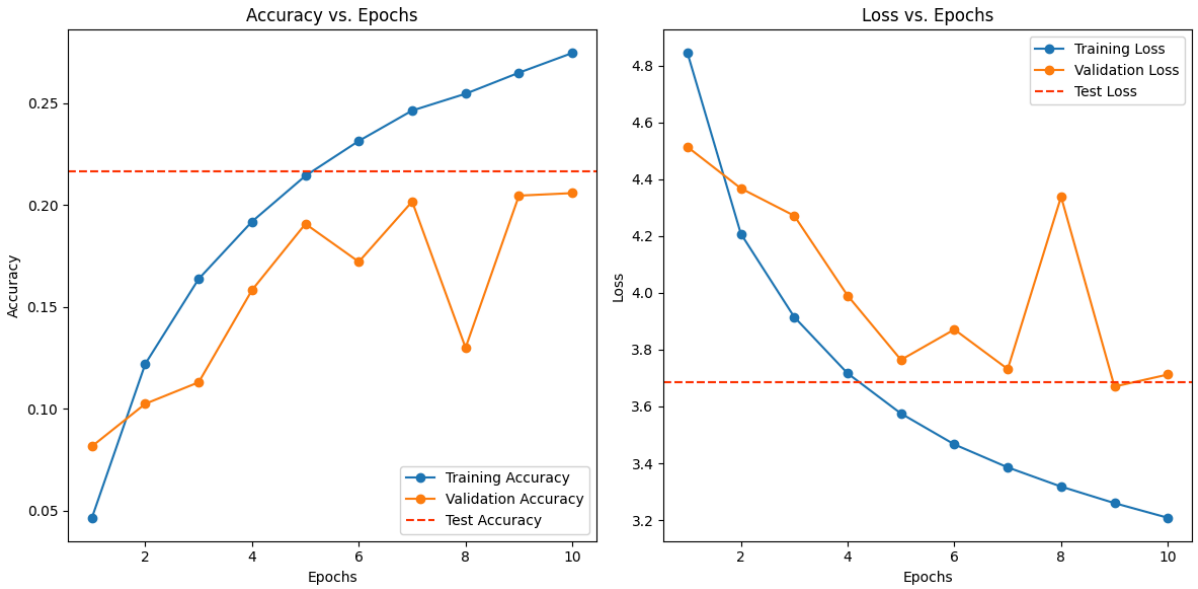


Χρόνος εκπαίδευσης: 6 λεπτά

- RMSProp με ρυθμό μάθησης 0.0005 και ορμή 0.5



Χρόνος εκπαίδευσης: 5,5 λεπτά

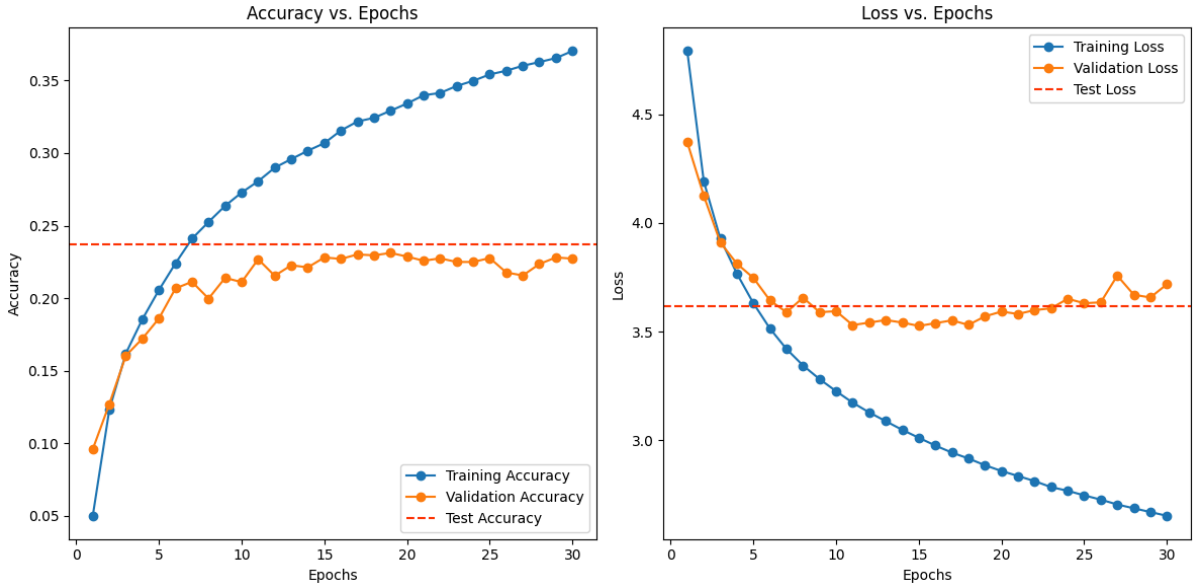


- RMSProp με ρυθμό μάθησης 0.0005 και ορμή 0.15

Χρόνος εκπαίδευσης: 6 λεπτά

Παρατηρώ ότι η ευστοχία του δικτύου δεν αυξήθηκε με τους βελτιστοποιητές και τις παραμέτρους που δοκιμάστηκαν. Αυτό μπορεί να σημαίνει είτε ότι ο Adam είναι πράγματι ο πιο αποτελεσματικός για το συγκεκριμένο δίκτυο, είτε ότι η

αρχιτεκτονική του δικτύου αναπτύχθηκε εκμεταλλευόμενη τις δυνατότητες του συγκεκριμένου βελτιστοποιητή και απλά κανένας άλλος δεν μπορεί να τον φτάσει και να τον ξεπεράσει. Καταληκτικά, επιλέγεται ο Adam ως βελτιστοποιητής με ρυθμό μάθησης 0.0005. Εκπαιδεύω το νευρωνικό δίκτυο για 30 εποχές για να βρω τη μέγιστη ευστοχία που μπορώ να πετύχω.



- Χρόνος εκπαίδευσης: 20 λεπτά
- Ευστοχία εκπαίδευσης: 37,74%
- Απώλεια εκπαίδευσης: 2,6184
- Ευστοχία επαλήθευσης: 22,72%
- Απώλεια επαλήθευσης: 3,7185
- Ευστοχία ελέγχου: 25,04%
- Απώλεια ελέγχου: 3,5363

Η ευστοχία του δικτύου σε νέες για αυτό εικόνες είναι 25,04%. Η ευστοχία επηρεάζεται από πολλούς παράγοντες, όπως την πολυπλοκότητα του σετ δεδομένων ή τον αριθμό κλάσεων. Εάν οι προβλέψεις γίνονταν τυχαία, η αναμενόμενη ακρίβεια θα ήταν $\frac{1}{number\ of\ classes} = \frac{1}{200} = 0{,}5\,\%$. Η επίτευξη 25% υπερβαίνει κατά πολύ το επίπεδο αναφοράς και δείχνει ότι το μοντέλο μαθαίνει σημαντικά μοτίβα.

Έλεγχος του νευρωνικού δικτύου

Έχοντας αποθηκεύσει το νεωρωνικό δίκτυο, θα κατηγοριοποιήσει εικόνες που δεν παρουσιάστηκαν σε κανένα από τα στάδια εκπαίδευσης, ελέγχου και επαλήθευσης. Είναι φυσικό οι εικόνες να είναι στις ίδιες διαστάσεις των εικόνων στις οποίες εκπαιδεύτηκε το δίκτυο ($64x64x3$).

$1^{η}$ εικόνα
κατηγοριοποίησης

Η εικόνα αυτή εμφανίζει μια πεταλούδα με τα χρώματα καφέ και μαύρο, έχοντας μαύρα μοτίβα, με πράσινο γρασίδι στο φόντο. Το δίκτυο την αντιστοίχισε ορθά στην κλάση 39, η οποία κλάση εμπεριέχει πεταλούδες. Το σετ δεδομένων εμπεριέχει εικόνες πεταλούδων ακριβώς ίδιων χρωμάτων. Επίσης, οι περισσότερες πεταλούδες έχουν τα ίδια μοτίβα στα φτερά τους και τα περισσότερα φόντο, όπως είναι φυσικό, έχουν πράσινο χρώμα, είτε γρασιδιού είτε λουλουδιών, το φυσικό περιβάλλον όπου βρίσκονται οι πεταλούδες. Το δίκτυο, λοιπόν, μπορεί να αφομοιώσει πολλά χαρακτηριστικά τα οποία εμφανίζονται επαναλαμβανόμενα στα δεδομένα εκπαίδευσης και να είναι βέβαιο στην απόφασή του.

$2^{η}$ εικόνα
κατηγοριοποίησης

Η εικόνα αυτή εμφανίζει ένα πορτοκάλι με εντελώς άσπρο φόντο. Το δίκτυο την αντιστοίχισε λανθασμένα στην κλάση 176, η οποία εμπεριέχει λαχανικά. Η κλάση 200 αποτελείται αποκλειστικά από πορτοκάλια, άρα η κατηγοριοποίηση έγινε εσφαλμένα. Στην κλάση 200 υπάρχουν φωτογραφίες με κομμένα πορτοκάλια και άλλες με πολύχρωμο και χωρίς μοτίβα φόντο. Δηλαδή, για τον ίδιο αριθμό εικόνων με την κλάση της πεταλούδας το δίκτυο αναμένεται να μάθει περισσότερα χαρακτηριστικά για την κλάση των πορτοκαλιών όπως το πως είναι κομμένα, πάνω σε δέντρα, σε τραπέζι κτλ. Στην κλάση 176 συναντώνται εικόνες λαχανικών, πολλές από τις οποίες είναι με εντελώς άσπρο φόντο. Επίσης συναντώνται λαχανικά με το ίδιο σχήμα και χρώμα της παραπάνω εικόνας. Επομένως, υπάρχουν χαρακτηριστικά της κλάσης 176 τα οποία εμφανίζονται στην εικόνα και για αυτό έγινε η συγκεκριμένη κατηγοριοποίηση.

Σύγκριση με άλλους Κατηγοριοποιητές

Στην ενδιάμεση εργασία συγκρίθηκαν οι κατηγοριοποιητές K-Nearest Neighbors για k=1 και 3 και Nearest Class Centroid. Η μέγιστη ευστοχία που είχε επιτευχθεί ήταν 4,1%, σε αντίθεση με το 25% αυτής της εργασίας. Παρατηρείται αύξηση σχεδόν 510%. Είναι λογική μία τέτοια σημαντική αύξηση, καθώς το δίκτυο αναγνωρίζει τις εικόνες από χαρακτηριστικά που έχει εξάγει, ενώ στους άλλους κατηγοριοποιητές γίνονταν υποθέσεις με μοναδικό κριτήριο τις τιμές των πίξελ των εικόνων. Οι τιμές αυτές συχνά εμπεριέχουν πληροφορία άσχετη με το κύριο αντικείμενο της εικόνας, όπως το περιβάλλον και το φόντο. Επίσης, τέτοιου είδους κατηγοριοποίηση είναι ευάλωτη στην φωτεινότητα, καθώς επηρρεάζει τις τιμές των πίξελ χωρίς να προσδίδει απαραίτητα επιπρόσθετη πληροφορία για την σωστή αντιστοίχιση σε κλάση.

Υλοποίηση

Παρακάτω θα περιγραφεί αναλυτικά ο κώδικας που χρησιμοποιήθηκε.

○ Σετ Δεδομένων

```
ds = load_dataset("zh-plus/tiny-imagenet")
```

• Φόρτωση των δεδομένων μέσω της βιβλιοθήκης $datasets$.

```
# Extract images and labels from database into variables
train_images = [ds['train'][i]['image'] for i in range(0, len(ds['train']))]
testing_images_original = [ds['valid'][i]['image'] for i in range(0, len(ds['valid']))]
```

```
train_labels = [ds['train'][i]['label'] for i in range(0, len(ds['train']))]
testing_labels_original = [ds['valid'][i]['label'] for i in range(0, len(ds['valid']))]
```

- Εξαγωγή των δεδομένων από την αρχική κλάση σε πίνακες.

```
# Split valid data to testing and valid images and labels
testing_images = [testing_images_original[i+j] for i in range(0,
len(testing_images_original), 50) for j in range(25)]
validation_images = [testing_images_original[i+j] for i in range(0,
len(testing_images_original), 50) for j in range(25, 50)]
testing_labels = [testing_labels_original[i+j] for i in range(0,
len(testing_labels_original), 50) for j in range(25)]
validation_labels = [testing_labels_original[i+j] for i in range(0,
len(testing_labels_original), 50) for j in range(25, 50)]
```

- Διαχωρισμός των labels και των εικόνων σε training και validation.
-

```
# Make images and labels into numpy arrays
train_images = [np.array(e) for e in train_images]
train_labels = [np.array(e) for e in train_labels]
testing_images = [np.array(e) for e in testing_images]
testing_labels = [np.array(e) for e in testing_labels]
validation_images = [np.array(e) for e in validation_images]
validation_labels = [np.array(e) for e in validation_labels]
```

- Μετατροπή των περιεχομένων των πινάκων σε arrays της βιβλιοθήκης $numpy$.

```
# Remove grayscale images from training set

indexes = []
for j in range(len(train_images)):
    if(len(train_images[j].shape)!=3):
        indexes.append(j)

for element in sorted(indexes, reverse=True):
    del train_images[element]
    del train_labels[element]


# Remove grayscale images from testing set

indexes = []
for j in range(len(testing_images)):
    if(len(testing_images[j].shape)!=3):
        indexes.append(j)

for element in sorted(indexes, reverse=True):
    del testing_images[element]
    del testing_labels[element]


# Remove grayscale images from validation set
```

```
for j in range(len(validation_images)):
    if(len(validation_images[j].shape)!=3):
        indexes.append(j)

for element in sorted(indexes, reverse=True):
    del validation_images[element]
    del validation_labels[element]
```

- Αφαίρεση των εικόνων με κωδικοποίηση grayscale έτσι ώστε να απομείνουν μόνο αυτές με κωδικοποίηση RGB.

```
# Make whole array into numpy array
train_images = np.array(train_images)
train_labels = np.array(train_labels)
testing_images = np.array(testing_images)
testing_labels = np.array(testing_labels)
validation_images = np.array(validation_images)
validation_labels = np.array(validation_labels)
```

- Μετατροπή των ίδιων των πινάκων σε arrays της βιβλιοθήκης $numpy$.

```
# Rescale image RGB values to [0, 1]
train_images = train_images / 255
testing_images = testing_images / 255
validation_images = validation_images / 255
```

- Κανονικοποίηση των τιμών RGB των εικόνων σε εύρος $(0,1)$.

○ Νευρωνικό Δίκτυο

Στη βιβλιοθήκη $keras$ της $python$ το μοντέλο δηλώνεται με τον εξής τρόπο:

```
model = models.Sequential()
```

Έπειτα προστίθενται με τη σωστή σειρά οι στρώσεις από τις οποίες θα αποτελείται το δίκτυο. Ακολουθεί η σύνταξη για όλα τα είδη που χρησιμοποιήθηκαν.

- Convolutional Layer

```
model.add(layers.Conv2D(32, (3,3), activation= 'relu', input_shape =
input_shape, strides=(2,2), padding='valid'))
```

Περνάται ως παράμετρος η στρώση ως μία από τις έτοιμες κλάσεις της βιβλιοθήκης, ακολουθούμενη από τις παραμέτρους με τη σειρά: αριθμός φίλτρων, kernel size, activation function, input shape (στη συγκεκριμένη περίπτωση $64x64x3$), strides και padding.

• Max-Pooling Layer

```
model.add(layers.MaxPooling2D((5, 5), strides=(1,1)))
```

Περνάται ως παράμετρος η στρώση ως μία από τις έτοιμες κλάσεις της βιβλιοθήκης, ακολουθούμενη από τις παραμέτρους με τη σειρά: pool size, strides.

• Flatten Layer

```
model.add(layers.Flatten())
```

Επιπεδώνει την είσοδο, δεν χρειάζεται παραμέτρους.

• Fully Connected (Dense) Layer

```
model.add(layers.Dense(128, activation='relu'))
```

Περνάται ως παράμετρος η στρώση ως μία από τις έτοιμες κλάσεις της βιβλιοθήκης, ακολουθούμενη από τις παραμέτρους με τη σειρά: number of neurons, activation function.

• Model Compilation

```
model.compile(optimizer='adam',  loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

Ρυθμίζει το μοντέλο για εκπαίδευση. Χρησιμοποιούνται οι ακόλουθοι παράμετροι: optimizer, loss, metrics (πληροφορίες που να εμφανίζονται στο τερματικό).

• Model Summary

```
model.summary()
```

Εμφανίζει μια περιγραφή του δικτύου στο τερματικό.

• Model Training

```
history = model.fit(train_images, train_labels, epochs=1,
validation_data=(validation_images, validation_labels),
shuffle=True)
```

Εκπαίδευση του μοντέλου. Χρησιμοποιούνται οι ακόλουθοι παράμετροι: training images, training labels, number of epochs, validation data (images and labels), shuffle (T/F). Τα δεδομένα αποθηκεύονται σε μεταβλητή γιατι χρησιμοποιούνται για την σχεδίαση των διαγραμμάτων.

• Model Timing

```
start_time = time.time()

end_time = time.time()
total_time = end_time - start_time
```

Χρονομέτρηση του μοντέλου. Το χρονόμετρο ξεκινάει πριν την εκπαίδευση του μοντέλου και σταματάει με την ολοκλήρωση του.

- Model Saving

```
model.save('30EpochImageClassifier.h5')
```

Αποθήκευση του μοντέλου ως αρχείο.

○ Αποτελέσματα

- Result Extraction

```
# Extract accuracy and loss
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
test_loss, test_accuracy = model.evaluate(testing_images,
testing_labels)
```

Αποθήκευση των αποτελεσμάτων ευστοχίας και απώλειας για τα στάδια της εκπαίδευσης, επαλήθευσης και ελέγχου.

- Accuracy Plot

```
# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(epochs, history.history['accuracy'], label='Training
Accuracy', marker='o')
plt.plot(epochs, history.history['val_accuracy'], label='Validation
Accuracy', marker='o')
plt.axhline(y=test_accuracy, color='r', linestyle='--', label='Test
Accuracy')
plt.title('Accuracy vs. Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc=4)
```

Δημιουργία του γραφήματος ευστοχίας, όπου σε ένα γράφημα υπάρχουν δύο καμπύλες και μία ευθεία της μορφής $y = a$. Όλες οι εντολές που αναγράφονται ανήκουν στην βιβλιοθήκη $matplotlib$.

- Loss Plot

```
# Loss plot
plt.subplot(1, 2, 2)
plt.plot(epochs, history.history['loss'], label='Training Loss',
marker='o')
plt.plot(epochs, history.history['val_loss'], label='Validation
Loss', marker='o')
plt.axhline(y=test_loss, color='r', linestyle='--', label='Test
Loss')
plt.title('Loss vs. Epochs')
plt.xlabel('Epochs')
```

```
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

Δημιουργία του γραφήματος απώλειας, όπου σε ένα γράφημα υπάρχουν δύο καμπύλες και μία ευθεία της μορφής $y = a$. Όλες οι εντολές που αναγράφονται ανήκουν στην βιβλιοθήκη $matplotlib$.

○ Έλεγχος

• Load model

```
# Load Model
loaded_model = keras.saving.load_model("/directory-path/model-name")
```

Αποκτά πρόσβαση στο μοντέλο από την τοπική τοποθεσία.

• Process image

```
# Process image
results = loaded_model.predict(img)
```

Επεξεργασία της εικόνας από το μοντέλο.

• Get predicted class

```
# Get the index of the highest probability (class label)
predicted_class = np.argmax(results, axis=1)

# Output the guessed classification
print("Predicted class:", predicted_class[0])
```

Εκτύπωση της κλάσης της κατηγοριοποίησης.