

# Νευρωνικά Δίκτυα - Βαθιά Μάθηση

3η Υποχρεωτική Εργασία  
Παπαδόπουλος Παναγιώτης 10697 HMMY

## Εισαγωγή

Το παρόν έργο αποσκοπεί στην ανάπτυξη ενός νευρωνικού δικτύου ικανού να παράγει εικόνες που αντιπροσωπεύουν τα αποτελέσματα απλών μαθηματικών πράξεων. Συγκεκριμένα, το δίκτυο λαμβάνει ως είσοδο τρεις εικόνες: δύο εικόνες ψηφίων από το γνωστό σύνολο δεδομένων *MNIST* και μία εικόνα που περιέχει το μαθηματικό σύμβολο της πράξης από το σύνολο δεδομένων *HASYv2*. Το δίκτυο εξάγει δύο εικόνες, οι οποίες αντιστοιχούν στα ψηφία του αποτελέσματος της πράξης. Το έργο βασίζεται σε αρχιτεκτονική *encoder – decoder*, όπου τα χαρακτηριστικά των εικόνων εισόδου κωδικοποιούνται σε μία λανθάνουσα αναπαράσταση (*latent representation*) και στη συνέχεια αποκωδικοποιούνται για να παραχθούν οι εικόνες εξόδου.

## Σετ Δεδομένων

Το σύνολο δεδομένων *MNIST* χρησιμοποιείται για την αναπαράσταση των αριθμητικών ψηφίων, καθώς περιέχει εικόνες ψηφίων  $28 \times 28$  σε ασπρόμαυρη κλίμακα. Για τα μαθηματικά σύμβολα, χρησιμοποιείται το *HASYv2*, το οποίο περιλαμβάνει εικόνες μαθηματικών συμβόλων σε ανάλυση  $32 \times 32$  και μετασχηματίζεται κατάλληλα ώστε να εναρμονιστεί με τη μορφή του *MNIST*. Ο μετασχηματισμός αυτός εμπεριέχει:

- Μετατροπή της διάστασης των εικόνων σε  $28 \times 28$  για να συμβαδίζει με του *MNIST*
- Αντιστροφή χρώματος, από μαύρους χαρακτήρες και λευκό φόντο σε λευκούς χαρακτήρες και μαύρο φόντο.

Το δίκτυο εκπαιδεύεται να εκτελεί τέσσερις βασικές πράξεις: πρόσθεση, αφαίρεση, πολλαπλασιασμό και διαιρεση. Κατά την εκπαίδευση, χρησιμοποιούνται δεδομένα εισόδου-στόχου, όπου τα αποτελέσματα των πράξεων χωρίζονται στα δύο ψηφία που τα απαρτίζουν. Αξίζει να σημειωθεί ότι:

- Η πράξη της αφαίρεσης έχει ως αποτέλεσμα την απόλυτη τιμή της διαφοράς των δύο αριθμών
- Η πράξη της διαιρεσης είναι στην πραγματικότητα ακέραια διαιρεση, τα δεκαδικά ψηφία δεν υπολογίζονται
- Σε περίπτωση διαιρεσης με το μηδέν, το αποτέλεσμα είναι επίσης μηδέν

Για την δημιουργία του σετ δεδομένων υπολογίζεται τυχαία μια μαθηματική εξίσωση που υπακούει τους παραπάνω κανόνες και τυχαία επιλέγονται εικόνες από τα αντίστοιχα προϋπάρχοντα σετ δεδομένων. Κάθε εικόνα κανονικοποιείται, με τις τιμές των πίξελ στο εύρος  $[0,1]$ .

## Εργαλεία

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

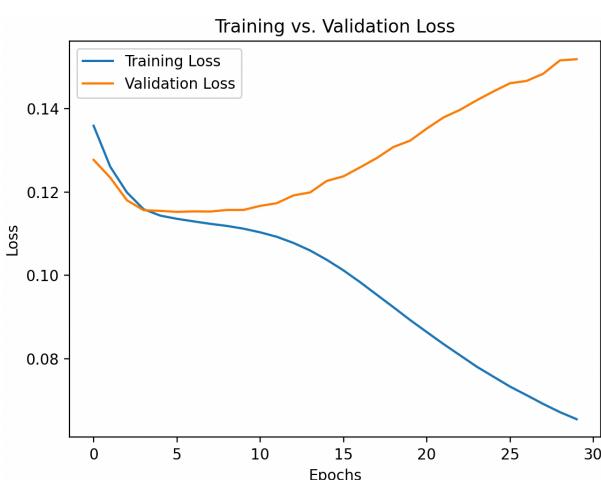
- *pimpy*, για την επεξεργασία των δεδομένων και την οργάνωση των εικόνων σε πολυδιάστατους πίνακες
- *deeplake*, για την πρόσβαση στο σύνολο δεδομένων *HASYv2*, το οποίο περιέχει τα μαθηματικά σύμβολα “+”, “-”, “x” και “/”.
- *opencv*, για την επεξεργασία των εικόνων από το *HASYv2*, ώστε να προσαρμοστούν στις διαστάσεις και την κανονικοποίηση του *MNIST*

## Ανάπτυξη του Δικτύου

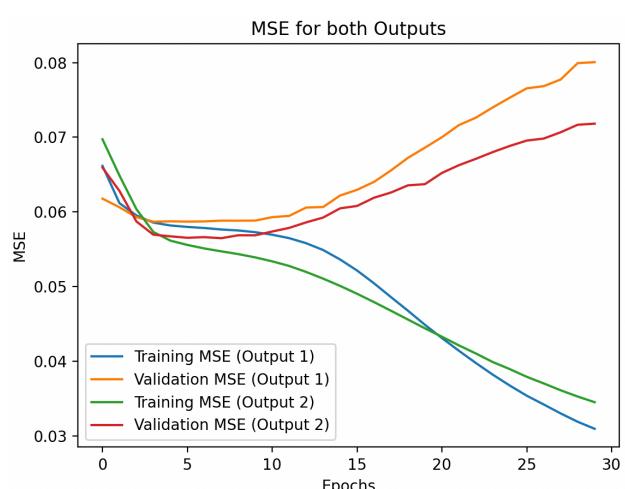
Για τον καθορισμό του βέλτιστου συνδυασμού υπερπαραμέτρων πραγματοποιηθήκαν αρχικές εκπαίδεύσεις δικτύων με τα παρακάτω κοινά χαρακτηριστικά:

- *3x3 kernel size*, ισορροπώντας ακρίβεια με απόδοση
- *2x2 pool size* για την μείωση της διάστασης του χώρου εξόδου
- *ReLU* συνάρτηση ενεργοποίησης, ιδανική για συνελικτικά νευρωτικά δίκτυα
- Λανθάνουσα μεγέθους 256
- *batch size* 32
- 30 εποχές
- Οι αρχικές εκπαίδεύσεις έγιναν με 50.000 αριθμό δειγμάτων

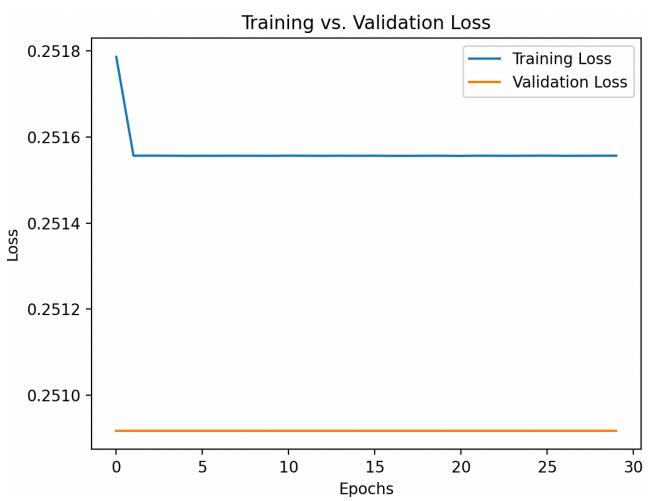
Για την αξιολόγηση των αποτελεσμάτων χρησιμοποιείται η απώλεια (*loss*) και *mean square error* κάθε εξόδου.



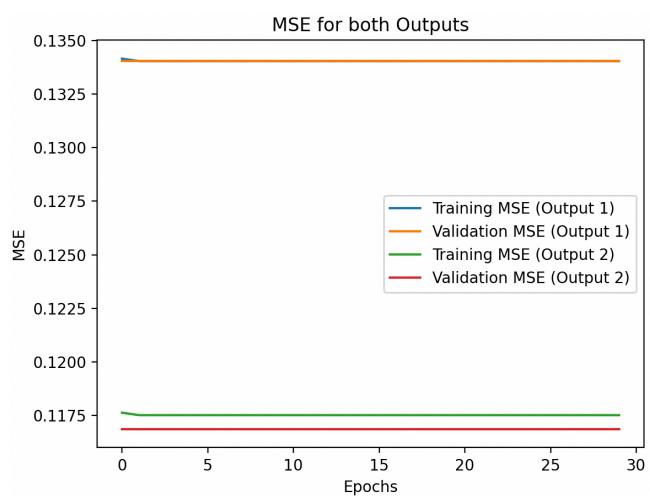
Απώλεια για 3 συνελικτικές, 3 αντίστροφες συνελικτικές στρώσεις, ρυθμό εκμάθησης 0.001



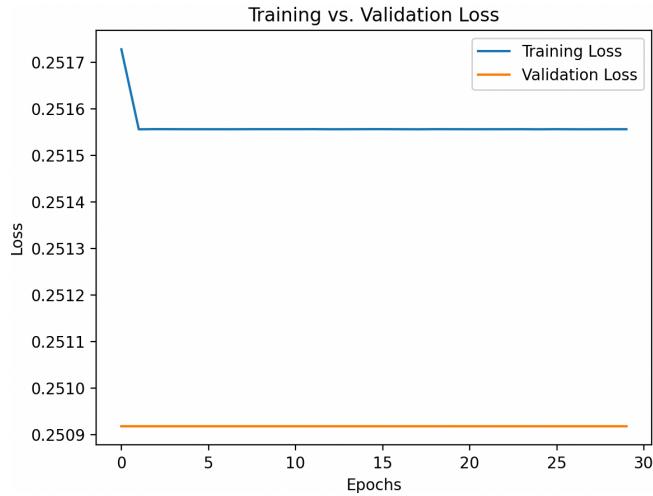
*MSE* για 3 συνελικτικές, 3 αντίστροφες συνελικτικές στρώσεις, ρυθμό εκμάθησης 0.001



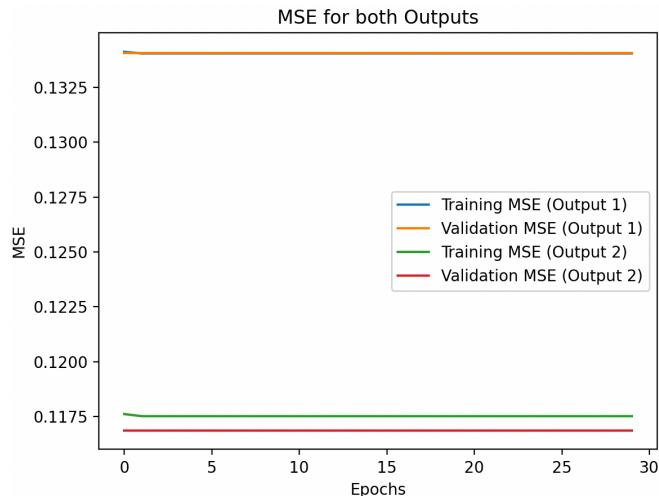
Απώλεια για 3 συνελικτικές, 3 αντίστροφες συνελικτικές στρώσεις, ρυθμό εκμάθησης 0.01



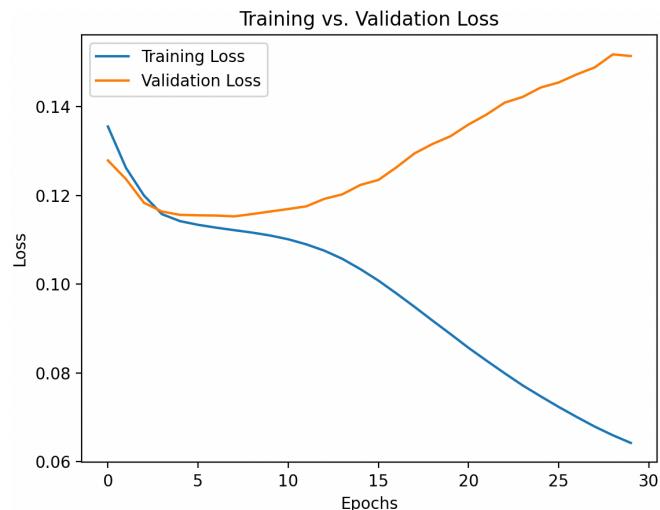
*MSE* για 3 συνελικτικές, 3 αντίστροφες συνελικτικές στρώσεις, ρυθμό εκμάθησης 0.01



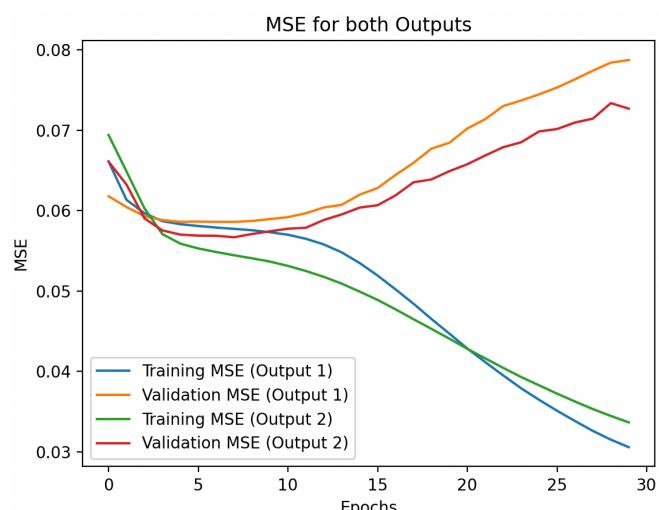
Απώλεια για 3 συνελικτικές, 3 αντίστροφες συνελικτικές στρώσεις, ρυθμό εκμάθησης 0.1



*MSE* για 3 συνελικτικές, 3 αντίστροφες συνελικτικές στρώσεις, ρυθμό εκμάθησης 0.1



Απώλεια για 3 συνελικτικές, 3 αντίστροφες συνελικτικές στρώσεις, ρυθμό εκμάθησης 0.001 και 100.000 δείγματα



*MSE* για 3 συνελικτικές, 3 αντίστροφες συνελικτικές στρώσεις, ρυθμό εκμάθησης 0.001 και 100.000 δείγματα

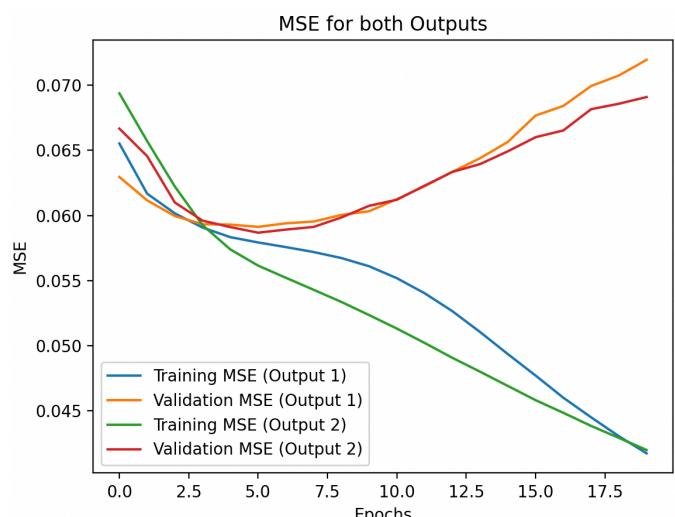
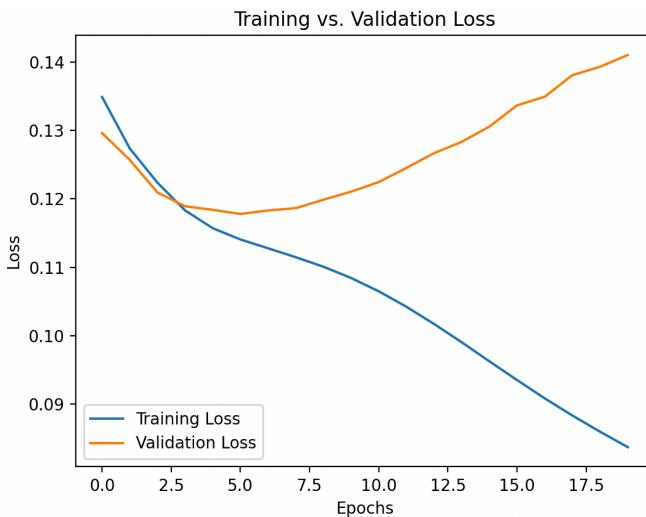
Μετά από χρόνο εκπαίδευσης 8 ωρών προκύπτουν οι παραπάνω κυματομορφές. Είναι εμφανές ότι το δίκτυο έχει την ικανότητα να μαθαίνει μόνο για ρυθμό εκμάθησης 0.001 από αυτούς που δοκιμάστηκαν. Επίσης, δεν υπάρχει φανερή διαφορά για την αύξηση του σετ δεδομένων. Για τα διαγράμματα *MSE* και των δύο εξόδων:

Η εκπαίδευση δείχνει συνεχή μείωση στο *MSE* για και τις δύο εξόδους, κάτι που υποδηλώνει ότι το μοντέλο μαθαίνει να παράγει πιο ακριβείς προβλέψεις με κάθε εποχή. Ωστόσο, το *validation MSE* (ειδικά για την πρώτη έξοδο) αρχίζει να αυξάνεται μετά από περίπου 10-15 εποχές, ενώ για τη δεύτερη έξοδο, η αύξηση είναι πιο έντονη.

Αυτή η αύξηση στο *validation MSE* είναι ένδειξη υπερπροσαρμογής (*overfitting*), όπου το μοντέλο αρχίζει να αποδίδει καλά στα δεδομένα εκπαίδευσης αλλά όχι στα δεδομένα επικύρωσης.

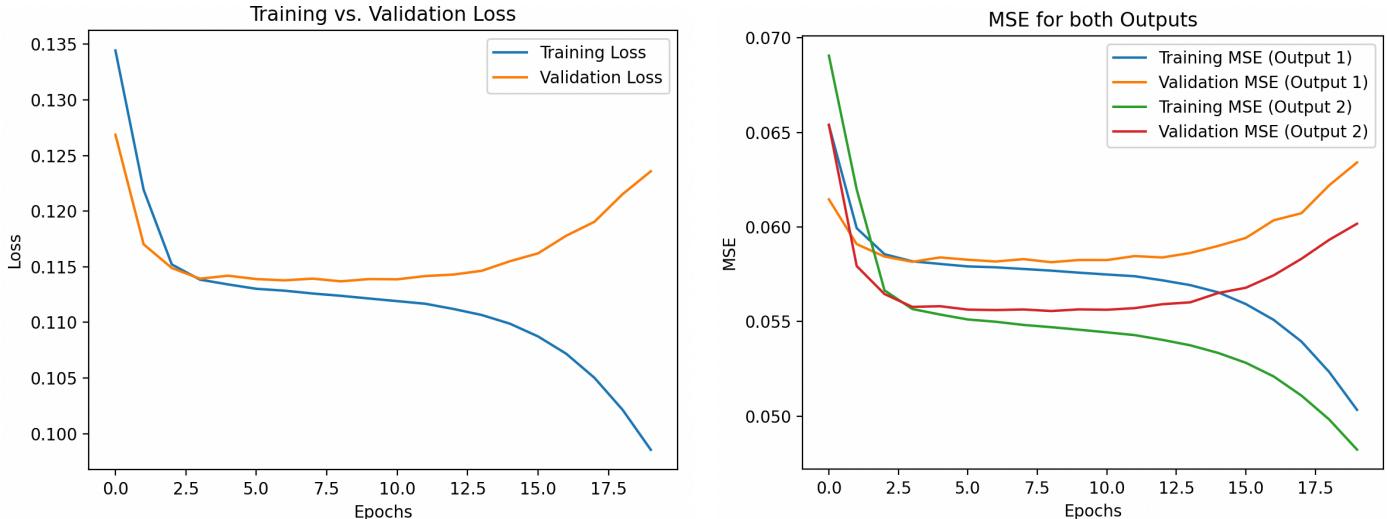
Από την άλλη, η εκπαίδευση έχει συνεχώς φθίνουσα απώλεια, γεγονός που σημαίνει ότι το μοντέλο βελτιστοποιεί τη συνάρτηση κόστους, ενώ η απώλεια επικύρωσης (*Validation Loss*) ακολουθεί το ίδιο μοτίβο με το *validation MSE*. Αυξάνεται μετά από περίπου 10 – 15 εποχές, υποδηλώνοντας το ίδιο πρόβλημα υπερπροσαρμογής.

Επομένως, διατηρείται ο ρυθμός εκμάθησης 0.001 και 50.000 στοιχεία για το σετ δεδομένων και δοκιμάζονται δύο νέα δίκτυα. Το πρώτο έχει αυξημένο αριθμό (4) συνελικτικών επιπέδων που επιτρέπουν στο δίκτυο να εξάγει πιο περίπλοκες ιεραρχικές ιδιότητες από τις εικόνες εισόδου. Επιπλέον, το μέγεθος του λανθάνοντος χώρου αυξάνεται από 256 σε 512, παρέχοντας έναν πιο εκφραστικό χώρο για την αναπαράσταση των πληροφοριών. Το *dense size* αναπροσαρμόστηκε σύμφωνα με τον αυξημένο αριθμό συνελικτικών επιπέδων και λειτουργιών *pooling*. Στη δεύτερη περίπτωση, ο αριθμός των συνελικτικών επιπέδων μειώθηκε (2), επιτρέποντας στο δίκτυο να μαθαίνει απλούστερες ιεραρχίες χαρακτηριστικών, μειώνοντας έτσι την πολυπλοκότητα της λανθάνουσας αναπαράστασης. Το μέγεθος του λανθάνοντος χώρου μειώθηκε σε 128 για να περιοριστεί η πολυπλοκότητα, ενώ το *dense size* αναπροσαρμόστηκε σύμφωνα με τον μειωμένο αριθμό συνελικτικών επιπέδων και λειτουργιών *pooling*. Τέλος, ο αριθμός των *transposed convolutional layers* μειώθηκε σε δύο, ώστε να ταιριάζει με τον απλούστερο κωδικοποιητή και τη λιγότερο λεπτομερή λανθάνουσα αναπαράσταση.



Απώλεια για 2 συνελικτικές, 2 αντίστροφες συνελικτικές στρώσεις, μέγεθος λανθάνουσας

*MSE* για 2 συνελικτικές, 2 αντίστροφες συνελικτικές στρώσεις, μέγεθος λανθάνουσας



Απώλεια για 4 συνελικτικές, 3 αντίστροφες συνελικτικές στρώσεις, μέγεθος λανθάνουσας  
512

*MSE* για 4 συνελικτικές, 3 αντίστροφες συνελικτικές στρώσεις, μέγεθος λανθάνουσας  
512

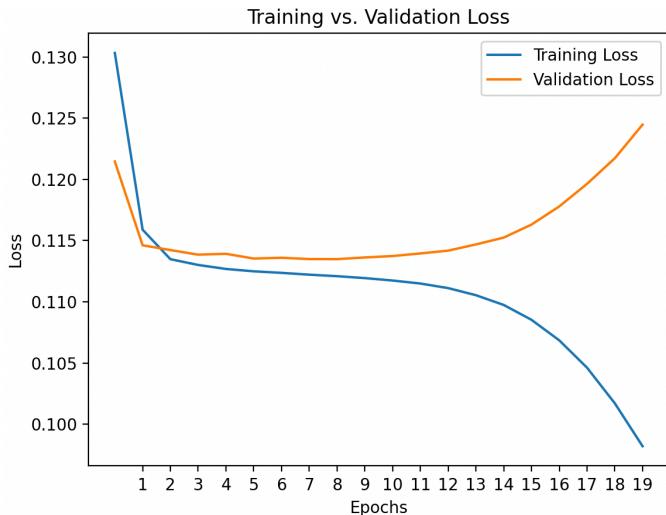
### Συνολικός χρόνος εκπαίδευσης: 2 ώρες

Η υψηλή τιμή του validation loss σε σχέση με το training loss δείχνει ότι το απλουστευμένο μοντέλο δεν έχει επαρκή δυνατότητα να μάθει τα χαρακτηριστικά των δεδομένων. Ο μικρός αριθμός επιπέδων και φίλτρων στις συνελικτικές στρώσεις έχει ως αποτέλεσμα να μην ανιχνεύονται πολύπλοκα μοτίβα στα δεδομένα.  
Το απλοποιημένο μοντέλο είναι ανεπαρκές για την επίλυση του προβλήματος. Ενώ μπορεί να μαθαίνει τα δεδομένα εκπαίδευσης (χαμηλό training loss), δεν μπορεί να γενικεύσει αποτελεσματικά στα δεδομένα επικύρωσης, με αποτέλεσμα υψηλότερο validation loss. Αυτό καθιστά αναγκαία τη χρήση πιο σύνθετων δομών.

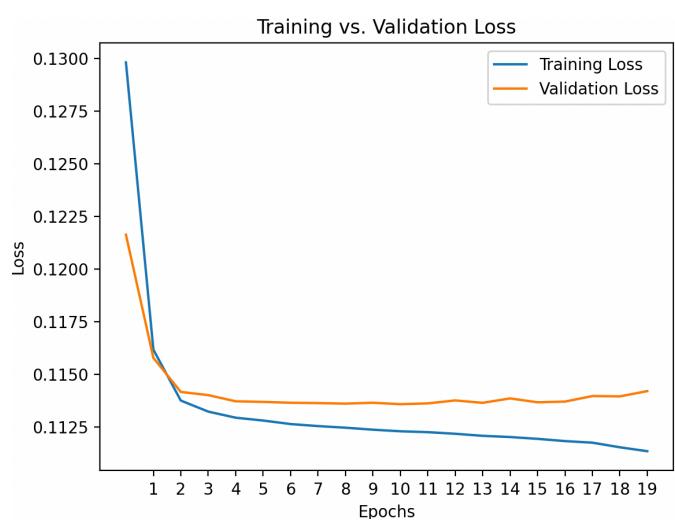
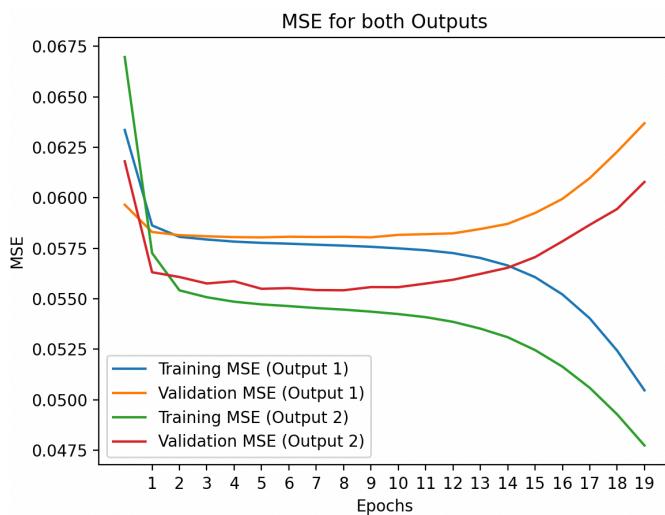
Το σύνθετο μοντέλο καταφέρνει να αποδώσει καλύτερα από τα άλλα μοντέλα στο validation set, γεγονός που υποδηλώνει ότι η αύξηση του αριθμού των επιπέδων και του μεγέθους του *latent space* βοήθησε στην αποδοτικότερη εκμάθηση χαρακτηριστικών.  
Παρόλα αυτά, η σχετικά υψηλότερη τιμή του *training loss* δείχνει ότι το μοντέλο δεν έχει μάθει πλήρως τα δεδομένα εκπαίδευσης, πιθανώς λόγω του μεγάλου αριθμού παραμέτρων που πρέπει να εκπαιδευτούν. Επομένως, διατηρείται αυτό το μοντέλο και γίνονται οι εξής περεταίρω αλλαγές:

- Αύξηση των δεδομένων σε 100.000 δείγματα
- Μείωση ρυθμού εκμάθησης
- Διαφοροποιήσεις στην διάσταση λανθάνουσας

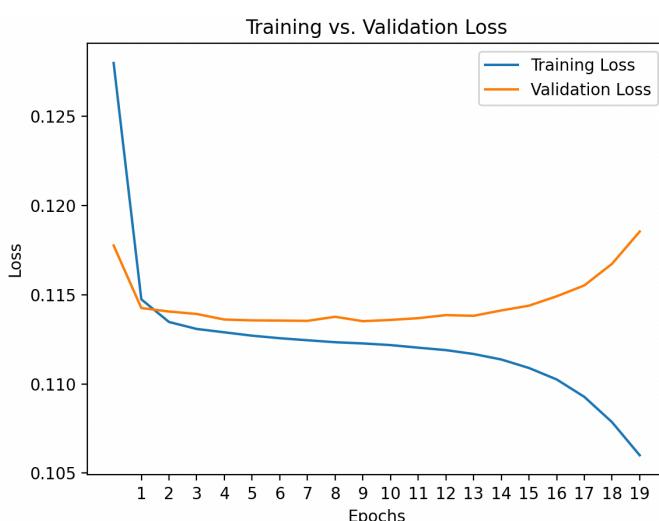
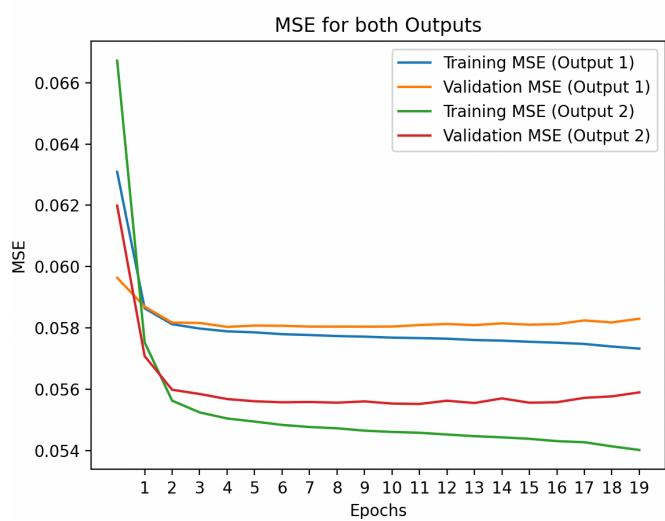
Μετά από 16 ώρες εκπαίδευσης:



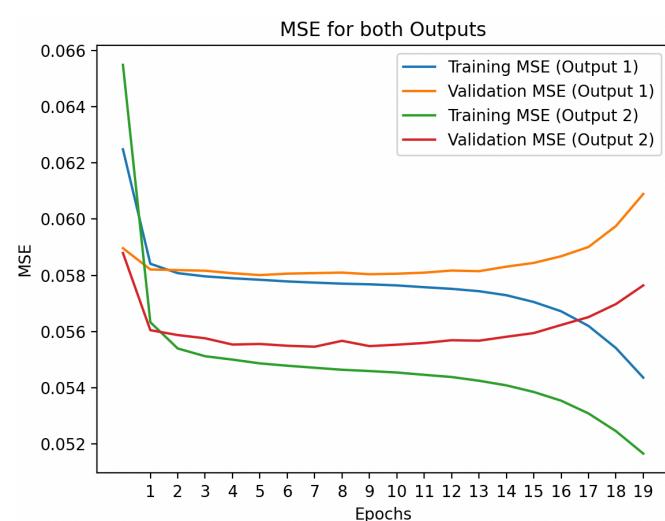
Απώλεια για 4 συνελικτικές, 3 αντίστροφες συνελικτικές στρώσεις, μέγεθος λανθάνουσας 512, ρυθμό εκμάθησης 0.0005



Απώλεια για 4 συνελικτικές, 3 αντίστροφες συνελικτικές στρώσεις, μέγεθος λανθάνουσας 128

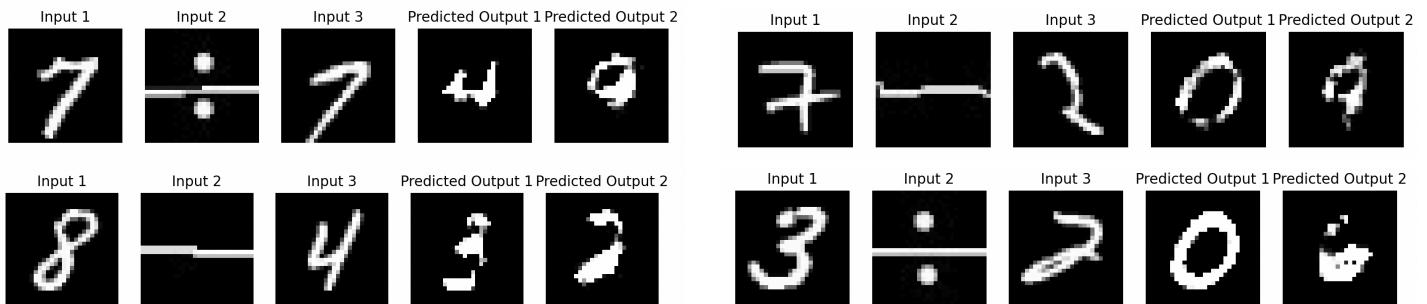


Απώλεια για 4 συνελικτικές, 3 αντίστροφες συνελικτικές στρώσεις, μέγεθος λανθάνουσας 1024

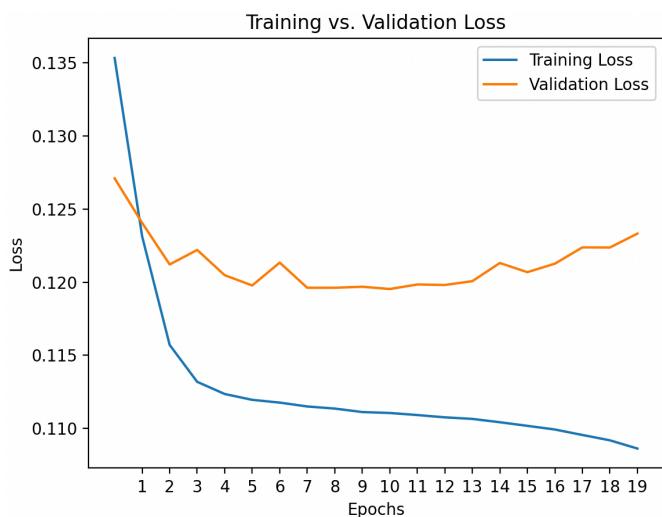


ΜΣΕ για 4 συνελικτικές, 3 αντίστροφες συνελικτικές στρώσεις, μέγεθος λανθάνουσας 1024

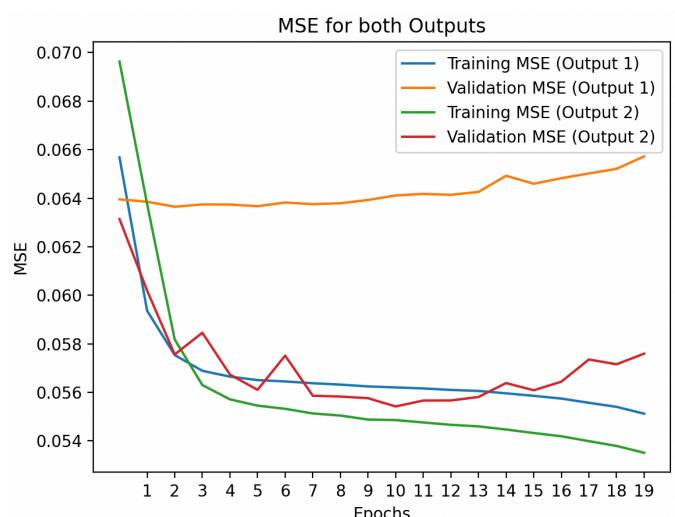
Είναι εμφανές ότι με μειωμένο ρυθμό εκμάθησης 0.0005 η απώλεια επαλήθευσης αρχίζει και πάλι να αυξάνεται, σε μικρότερο, ωστόσο, βαθμό. Το ίδιο συμβαίνει και για την αύξηση της λανθάνουσας σε 1024. Όμως, για μείωση της λανθάνουσας σε 128 φαίνεται να σταθεροποιείται η απώλεια και το *mean square error* κάθε εξόδου. Με δοκιμές χρησιμοποιώντας νέα δεδομένα από αυτά της εκπαίδευσης και επικύρωσης, παρατηρείται ότι το δίκτυο για πολλούς συνδυασμούς αριθμών θα εμφάνιζε σωστό αποτέλεσμα, αν ο τελεστής ήταν διαφορετικός.



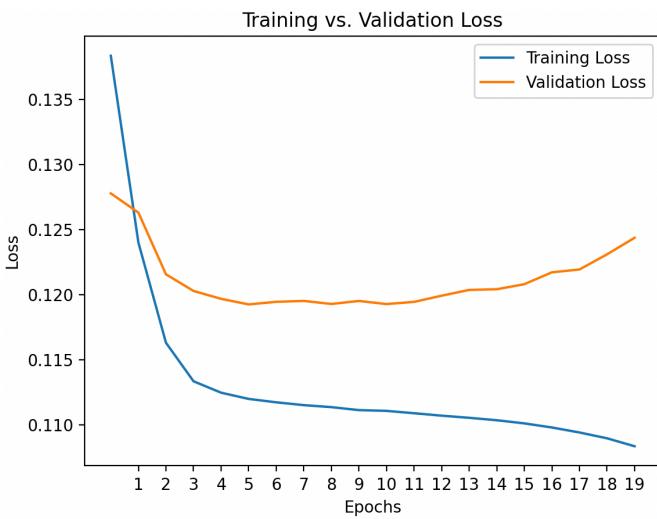
Επομένως, θα δοκιμαστεί η εισαγωγή βάρους στην είσοδο των δεδομένων της εικόνας του τελεστή. Μετά από 6 ώρες προκύπτουν οι μορφές:



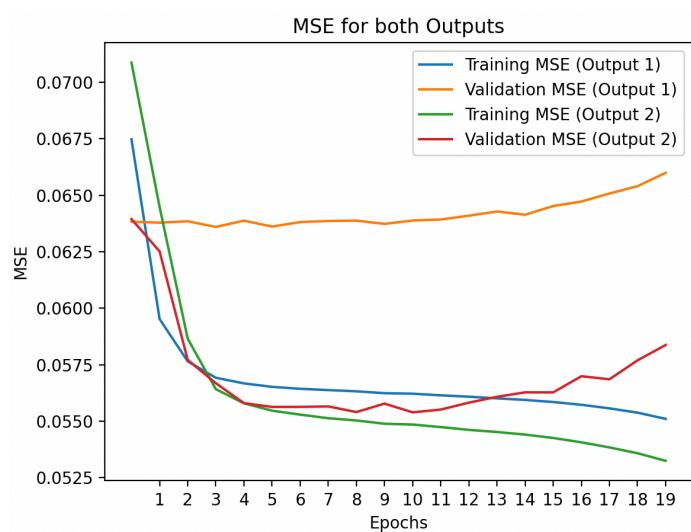
Απώλεια για συντελεστή βάρους 0.8



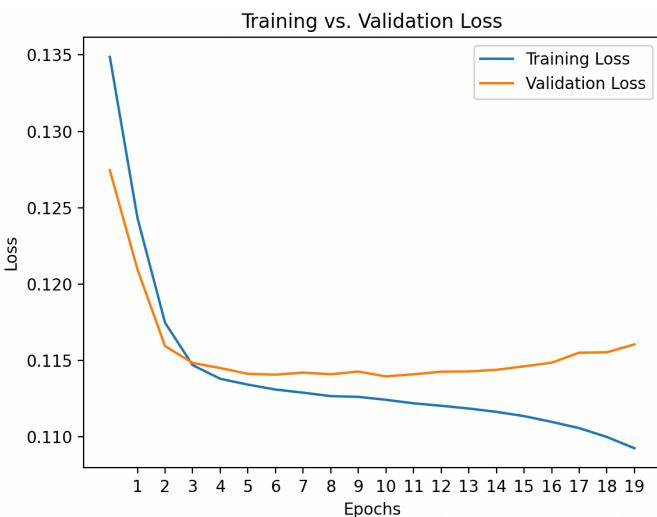
MSE για συντελεστή βάρους 0.8



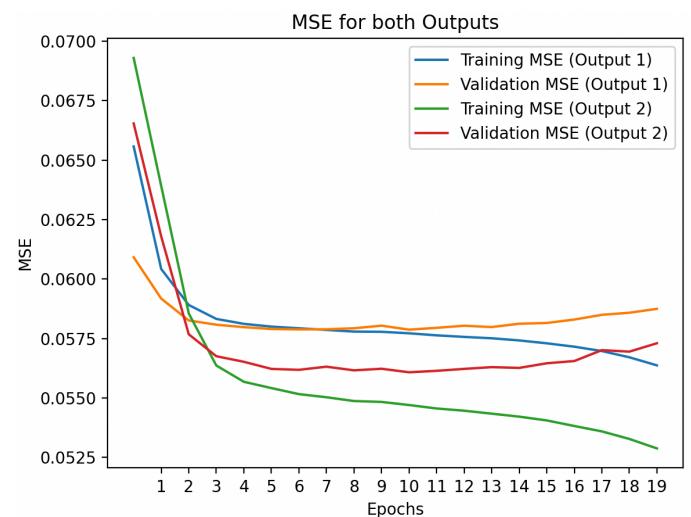
Απώλεια για συντελεστή βάρους 1.2



Απώλεια για συντελεστή βάρους 1.2

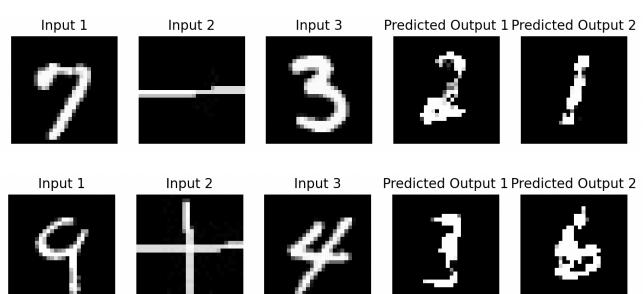
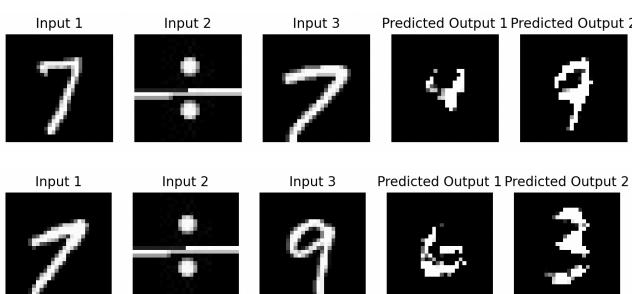


Απώλεια για συντελεστή βάρους 2

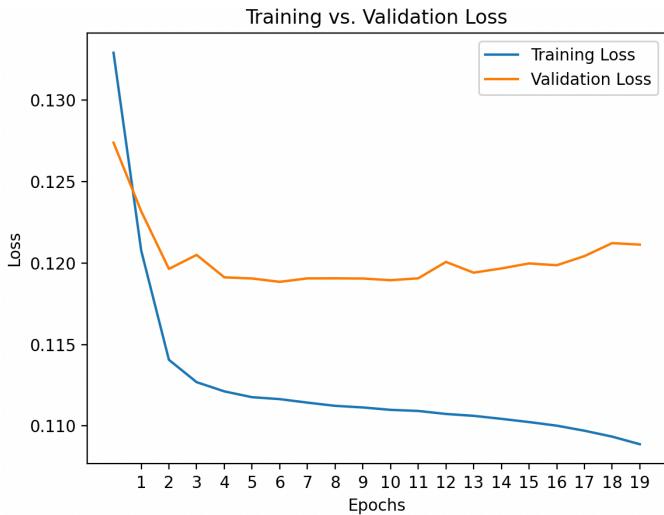


MSE για συντελεστή βάρους 2

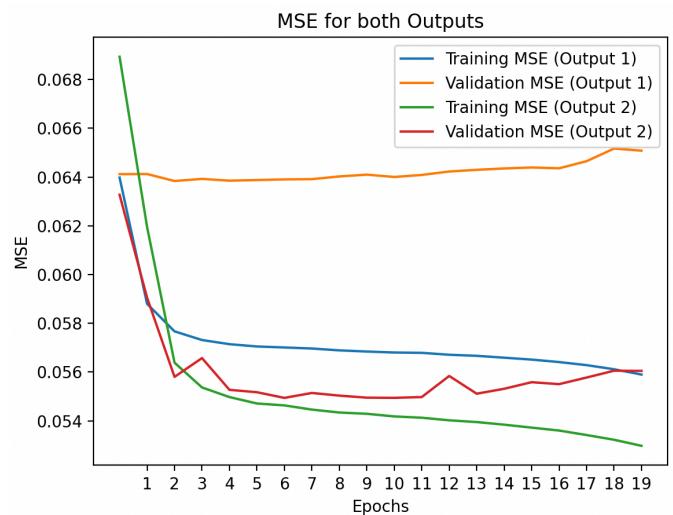
Αυτές οι τεχνικές δεν λειτούργησαν αφού παρατηρείται αύξηση της απώλειας συγκριτικά με το προηγούμενο μοντέλο. Με παραπάνω ελέγχους φαίνεται η πράξη του πολλαπλασιασμού να κυριαρχεί στα αποτελέσματα, ακόμη κι όταν δεν θα έπρεπε:



Με μείωση των δειγμάτων πολλαπλασιασμού στο σετ δεδομένων και μετά από 4 ώρες προκύπτει:

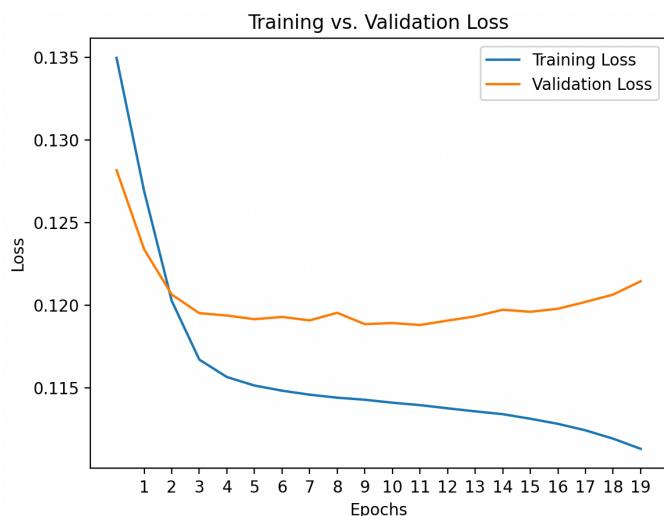


Απώλεια για αναλογία δειγμάτων πολ/μού 1 / 3

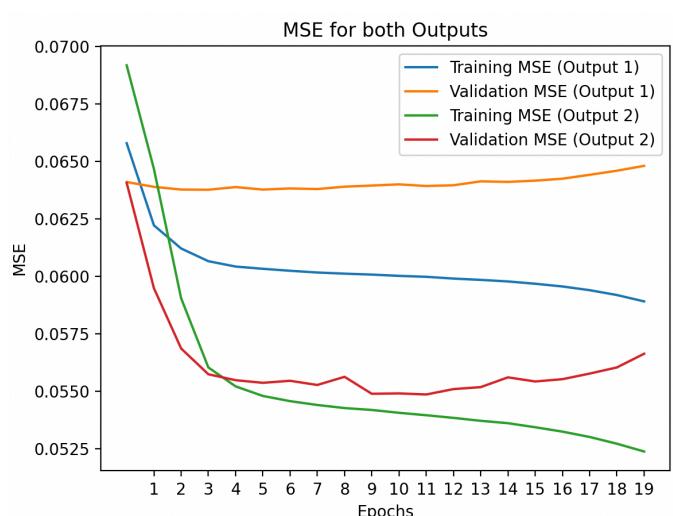


*MSE* για αναλογία δειγμάτων πολ/μού 1 / 3

Μετά από δοκιμές φαίνεται να κυριαρχεί η πράξη της πρόσθεσης, οπότε αφαιρούνται δείγματα και από αυτή την πράξη και μετά από 3 ώρες προκύπτει:



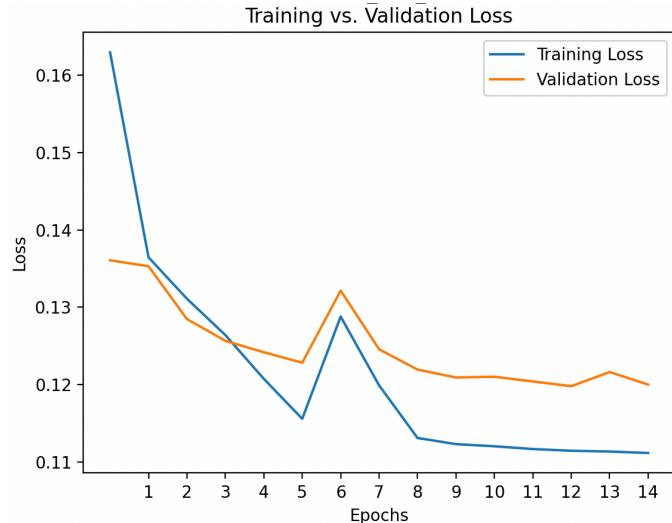
Απώλεια για αναλογία δειγμάτων πολ/μού και πρόσθεσης 1 / 3



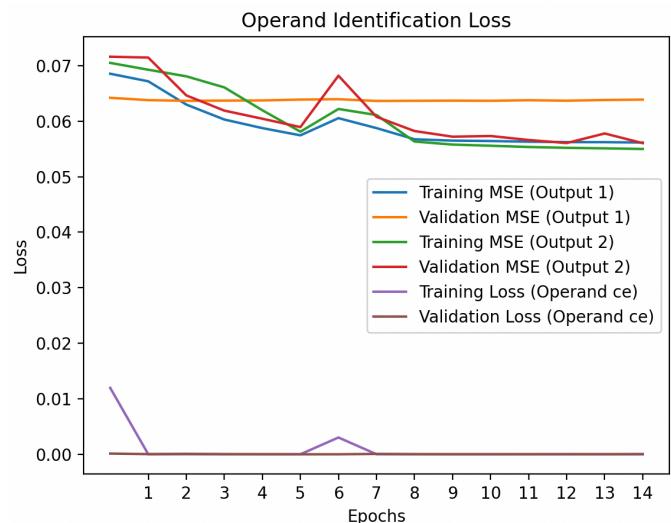
*MSE* για αναλογία δειγμάτων πολ/μού και πρόσθεσης 1 / 3

Είναι εμφανές ότι οι τεχνικές αυτές δεν συμβάλουν στην μείωση της απώλειας. Επίσης, οι εικόνες εξόδου ήταν χαμηλότερες ποιότητας λόγω των περιορισμένων δειγμάτων. Εικάζεται ότι το φαινόμενο κατά το οποίο το νευρωνικό δίκτυο προτιμά να εκτελεί πράξεις πολλαπλασιασμού, ακόμη και όταν οι εισαγόμενες πράξεις είναι διαφορετικές, μπορεί να εξηγηθεί από τον τρόπο που υπολογίζεται η απώλεια και τη φύση των αριθμητικών πράξεων. Ο πολλαπλασιασμός παράγει γενικά μεγαλύτερα αποτελέσματα από τις υπόλοιπες πράξεις, όπως η πρόσθεση ή η αφαίρεση. Αυτό έχει ως αποτέλεσμα η απώλεια, που βασίζεται στη μέση τετραγωνική απόκλιση (*Mean Squared Error – MSE*), να ευνοεί μικρότερες αποκλίσεις όταν η πρόβλεψη είναι πιο κοντά σε αποτελέσματα πολλαπλασιασμού.

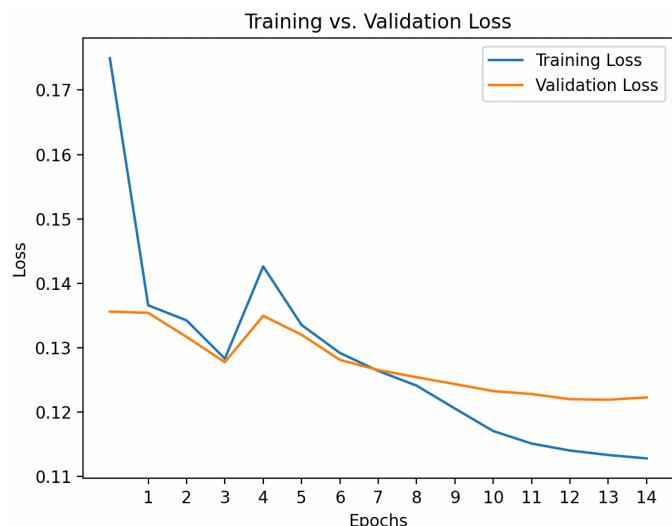
Επομένως, θα δημιουργηθεί νέα έξοδος του δικτύου για τον τελεστή της πράξης και θα δοκιμαστεί η εισαγωγή βαρών στην απώλεια για να δοθεί μεγαλύτερη σημασία στην πρόβλεψη του τελεστή.



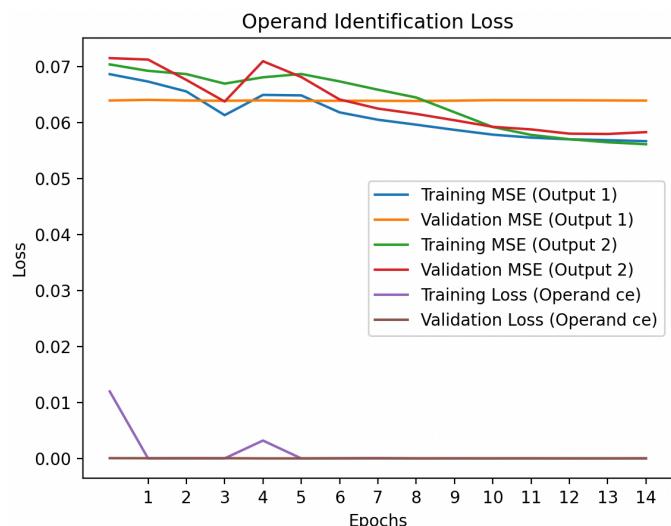
Απώλεια για βάρος τελεστή 2



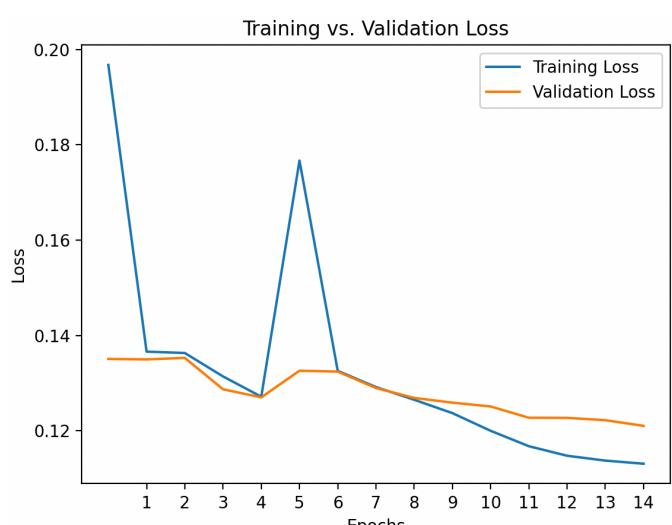
Απώλεια για κάθε έξοδο με βάρος τελεστή 2



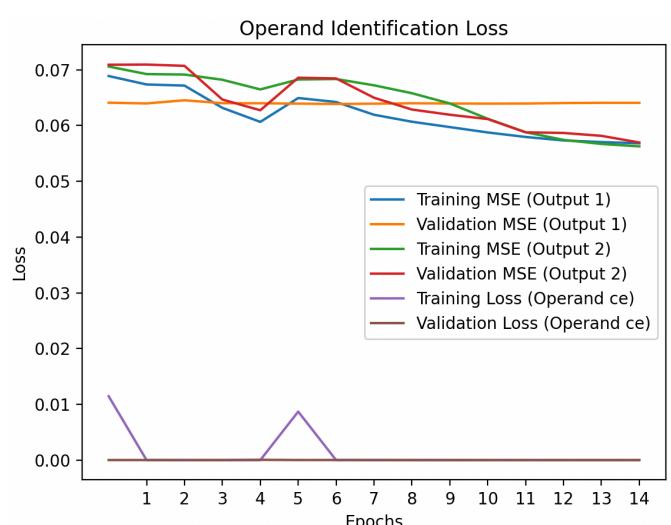
Απώλεια για βάρος τελεστή 3



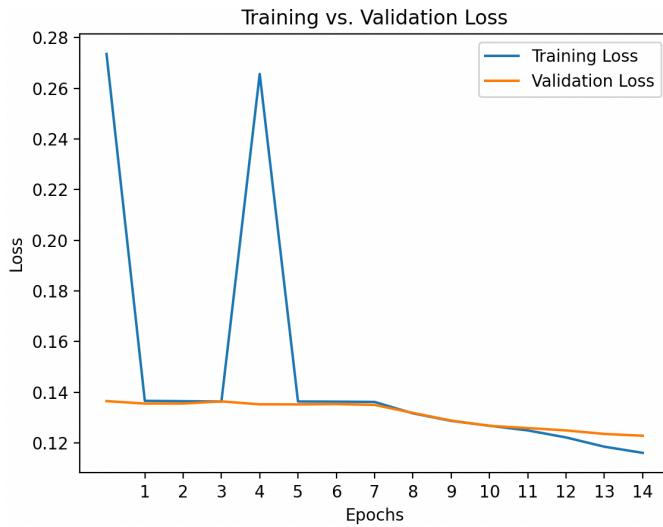
Απώλεια για κάθε έξοδο με βάρος τελεστή 3



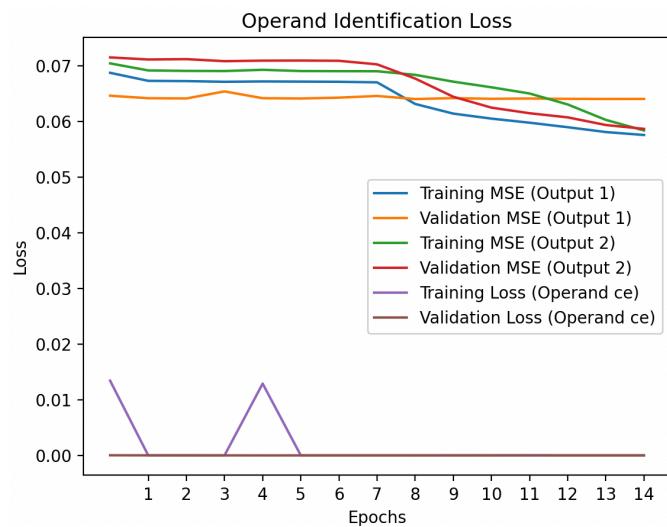
Απώλεια για βάρος τελεστή 5



Απώλεια για κάθε έξοδο με βάρος τελεστή 5

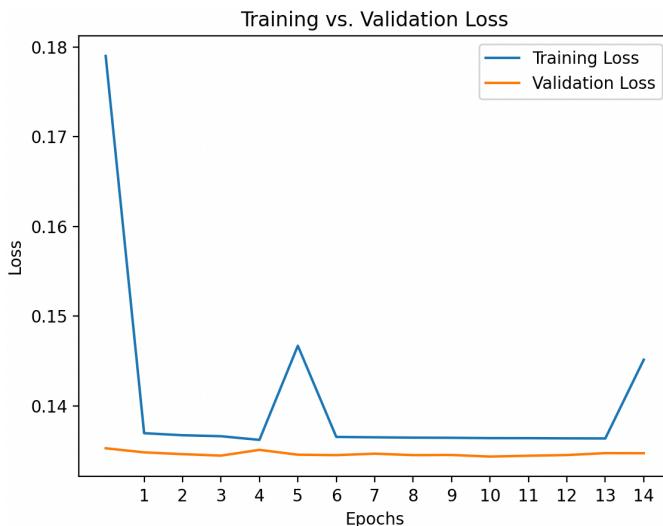


Απώλεια για βάρος τελεστή 10

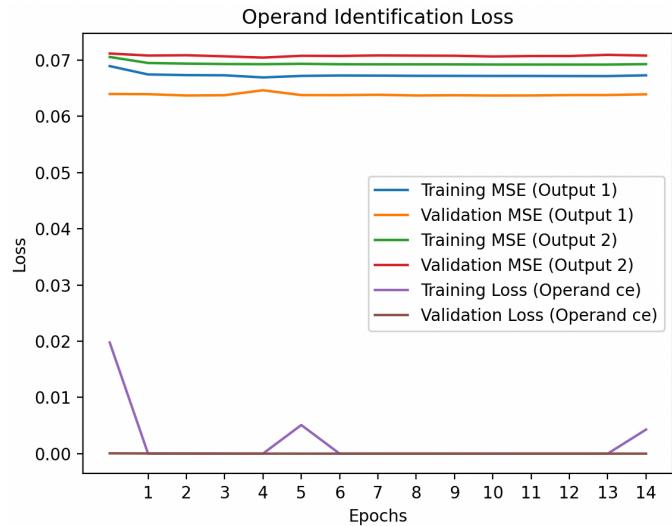


Απώλεια για κάθε έξοδο με βάρος τελεστή 10

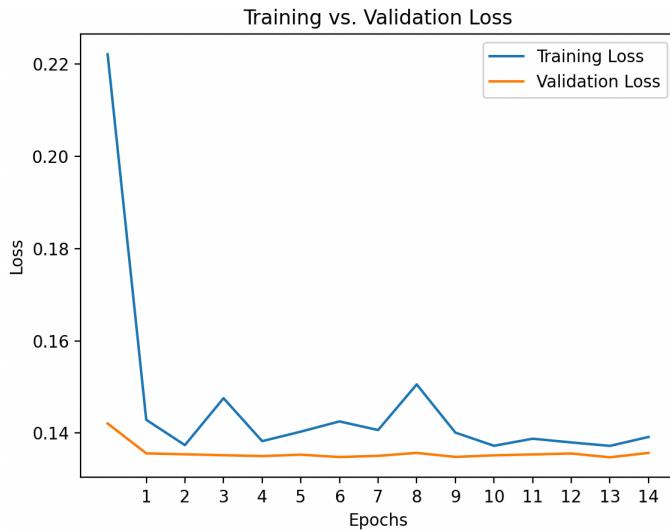
Φαίνεται το ψηφίο των μονάδων να συνεχίζει να μειώνει την απώλεια του, ενώ η απώλεια τελεστή εμφανίζεται μηδενική, γεγονός το οποίο στον έλεγχο δεν ισχύει. Από τις παραπάνω εκπαιδεύσεις, καλύτερες επιδόσεις είχε η απώλεια στην αναγνώριση τελεστή με βάρος 2, τόσο στα διαγράμματα όσο και σε δοκιμές ελέγχου. Τελικά, θα δοκιμαστούν στρώσεις *Dropout* και έπειτα *batch normalization* ως τελικές απόπειρες για μείωση της απώλειας.



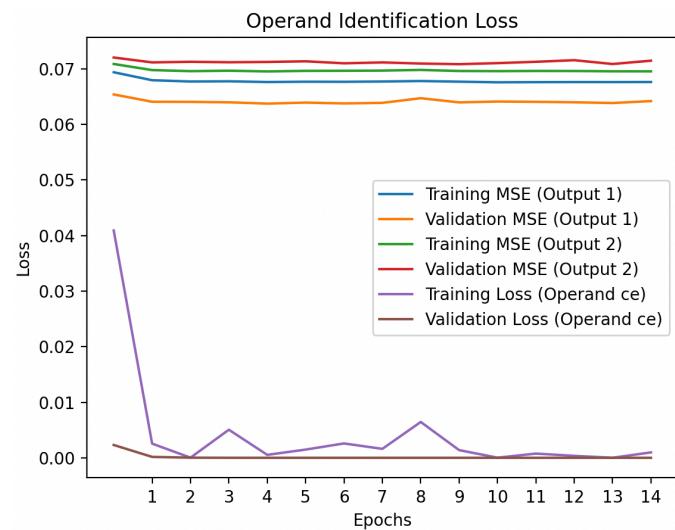
Απώλεια με *Dropout Rate* 0.2



Απώλεια για κάθε έξοδο με *Dropout Rate* 0.2



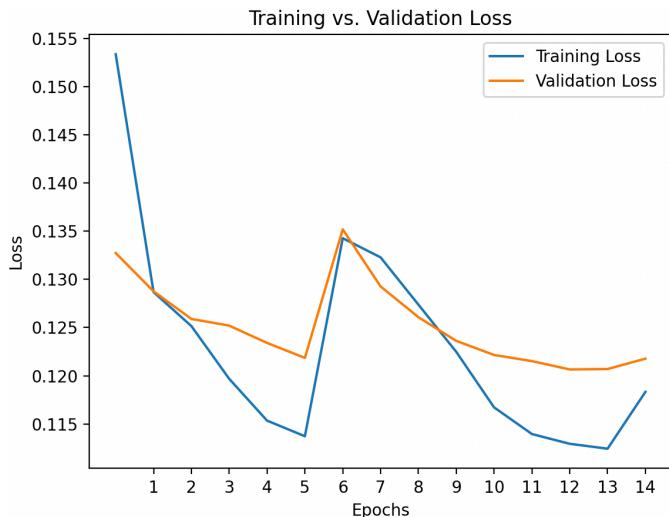
Απώλεια με *Dropout Rate 0.5*



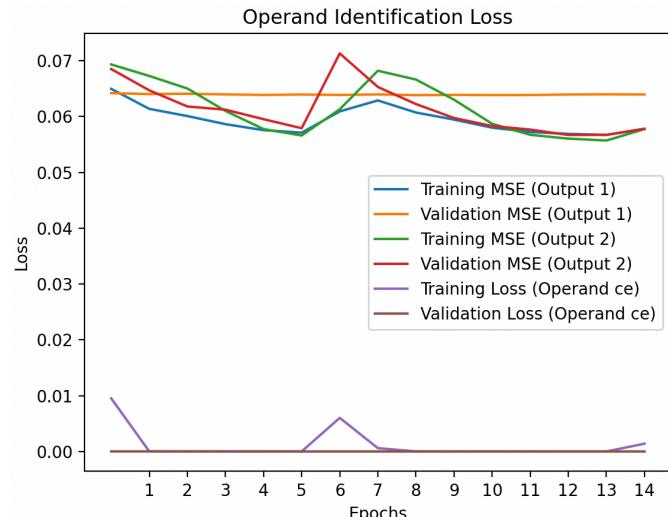
Απώλεια για κάθε έξοδο με  
*Dropout Rate 0.5*

Διάρκεια εκπαίδευσης: 4 ώρες

Η εισαγωγή στρώσεων *Dropout* δεν ευνόησε το δίκτυο, αντιθέτως είχε αρνητική επιρροή στις εικόνες εξόδου. Για *Batch Normalization*:



Απώλεια με *Batch Normalization*



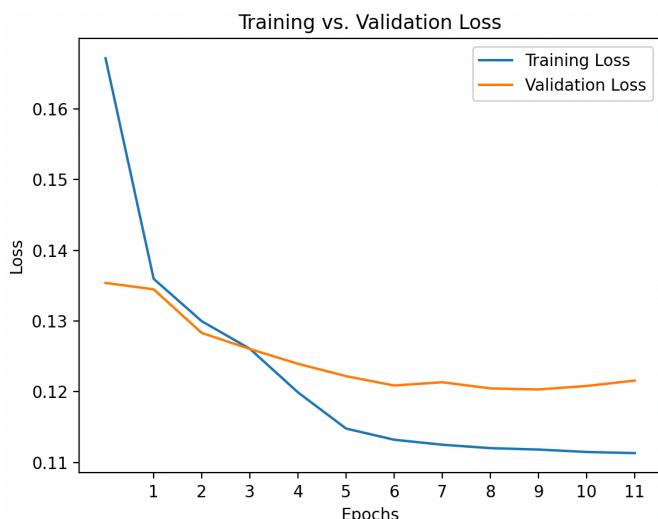
Απώλεια για κάθε έξοδο με  
*Batch Normalization*

Διάρκεια εκπαίδευσης: 1.5 ώρα

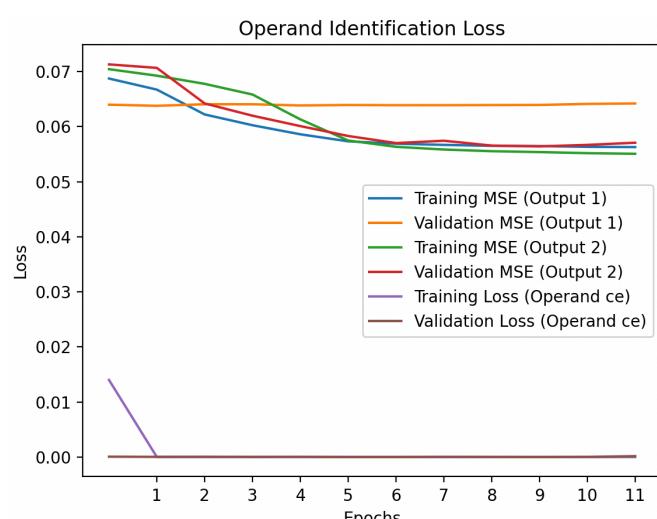
Συγκρίνοντας όλες τις αρχιτεκτονικές δικτύων για το πρότζεκτ, κρίνεται ότι τις καλύτερες επιδόσεις έχει αυτό με τα εξής χαρακτηριστικά:

- 4 συνελικτικές στρώσεις
- 3 αντίστροφες συνελικτικές στρώσεις
- Μέγεθος λανθάνουσας 126
- Ρυθμός Εκμάθησης 0.001
- Διατηρείται η έξιδος αναγνώρισης του τελεστή με συντελεστή βάρους 2

Επομένως, εκπαιδεύεται το δίκτυο με τα παραπάνω χαρακτηριστικά μέχρι να ξεκινήσει να εμφανίζεται το φαινόμενο της υπερπροσαρμογής. Άρα για 12 εποχές με 1.5 ώρα εκπαίδευσης τα τελικά αποτελέσματα είναι:



Απώλεια

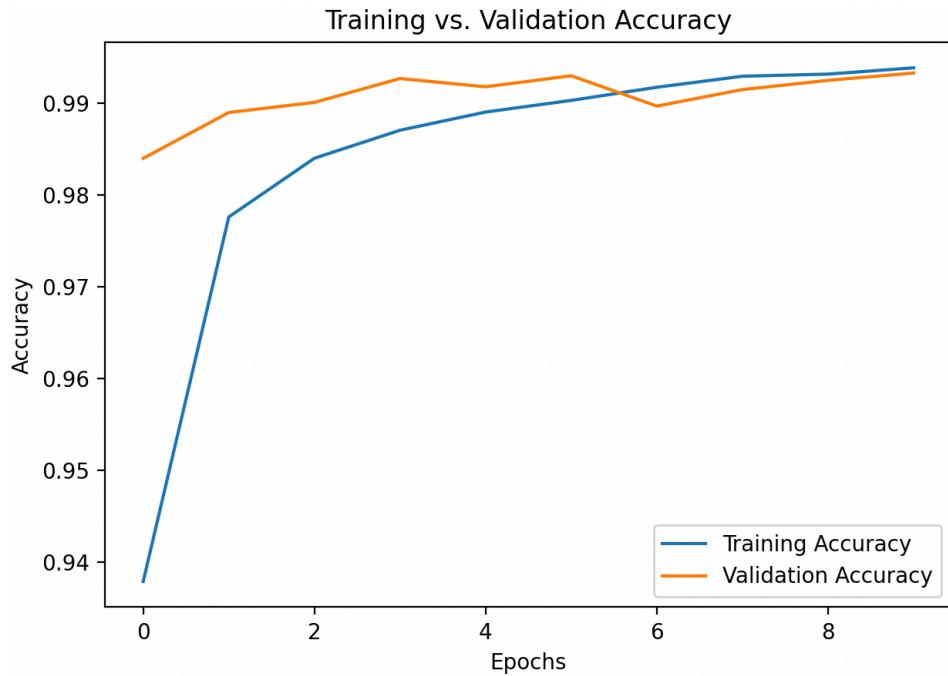


Απώλεια για κάθε έξιδο

## Έλεγχος του Δικτύου

Θα αναπτυχθεί ένα δίκτυο κατηγοριοποίησης εικόνων *MNIST* για την διερεύνηση της ποιότητας των εικόνων εξόδου των αριθμητικών πράξεων.

Το δίκτυο αυτό αποτελείται από μία συνελικτική στρώση 32 φίλτρων, μία *MaxPooling* με μέγεθος *pool*  $2 \times 2$ , μία συνελικτική στρώση 64 φίλτρων, μία ίδια *MaxPooling* και μία πλήρης συνδεδεμένη στρώση με 128 νευρώνες. Το δίκτυο αυτό μετά από 15 λεπτά και 10 εποχές, είχε τα παρακάτω αποτελέσματα.



Ευστοχία του δικτύου κατηγοριοποίησης εικόνων *MNIST*

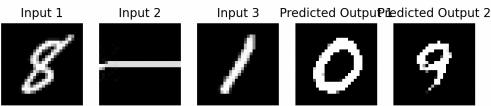
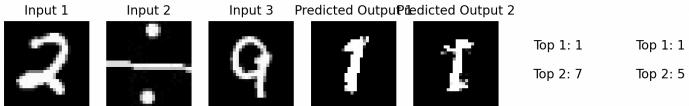
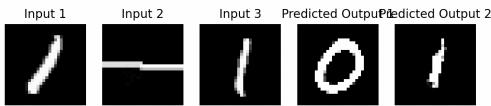
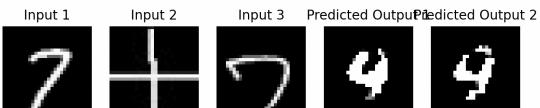
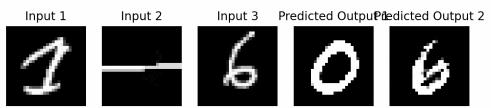
Επομένως, τώρα μπορούν να αξιολογηθεί η επίδοση του αρχικού δικτύου.

Τα αποτελέσματα χωρίζονται σε τρεις κατηγορίες. Στην πρώτη κατηγορία ανήκουν τα ζεύγη εικόνων που απεικονίζουν το σωστό αποτέλεσμα της μαθηματικής πράξης.

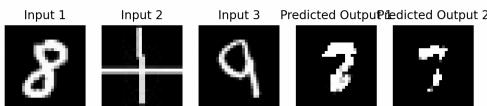
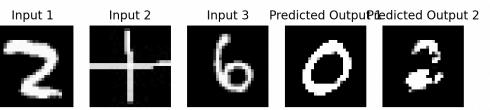
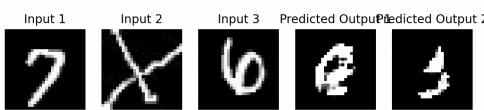
Input 1	Input 2	Input 3	Predicted Output	Predicted Output 2	Top 1: 2	Top 1: 1	Top 2: 3	Top 2: 5	Input 1	Input 2	Input 3	Predicted Output	Predicted Output 2	Top 1: 0	Top 1: 9	Top 2: 6	Top 2: 8
					Top 1: 2	Top 1: 1	Top 2: 3	Top 2: 5						Top 1: 0	Top 1: 9	Top 2: 6	Top 2: 8
					Top 1: 0	Top 1: 5	Top 2: 6	Top 2: 3						Top 1: 0	Top 1: 2	Top 2: 6	Top 2: 8
					Top 1: 0	Top 1: 3	Top 2: 6	Top 2: 5						Top 1: 0	Top 1: 0	Top 2: 6	Top 2: 6

Στην δεύτερη κατηγορία ανήκουν τα ζεύγη εικόνων τα οποία απεικονίζουν αποτέλεσμα διαφορετικής αριθμητικής πράξης από αυτήν που υποδηλώνει η εικόνα του τελεστή.

Input 1	Input 2	Input 3	Predicted Output	Predicted Output 2	Top 1: 0	Top 1: 4	Top 2: 9	Top 2: 5	Input 1	Input 2	Input 3	Predicted Output	Predicted Output 2	Top 1: 0	Top 1: 3	Top 2: 6	Top 2: 5
					Top 1: 0	Top 1: 4	Top 2: 9	Top 2: 5						Top 1: 0	Top 1: 3	Top 2: 6	Top 2: 5
					Top 1: 0	Top 1: 3	Top 2: 6	Top 2: 5						Top 1: 0	Top 1: 0	Top 2: 6	Top 2: 8



Τέλος, στην τρίτη κατηγορία τα αποτελέσματα είναι λανθασμένα για οποιαδήποτε αριθμητική πράξη.



Συγκρίνοντας τις εικόνες με την κατηγοριοποίηση του δικτύου αναγνώρισης των ψηφίων είναι εμφανές ότι η κατηγοριοποίηση στις περισσότερες περιπτώσεις γίνεται σωστά, ενώ οι ίδιες οι εικόνες εξόδου όντως αντιπροσωπεύουν ψηφία αντίστοιχα του σετ δεδομένων *MNIST*. Το προβληματικό σημείο δεν είναι η δημιουργία των εικόνων ούτε η αντιστοίχιση τους σε ψηφίο, αλλά να αντιπροσωπεύουν τη σωστή πράξη.

## Κώδικας

Το αρχείο *MNIST\_Image\_multiple\_operations.py* είναι υπεύθυνο για την παραγωγή του σετ δεδομένων και την εκπαίδευση του δικτύου παραγωγής εικόνων:

### 1. Φόρτωση Δεδομένων

Αυτό το τμήμα διαχειρίζεται τη φόρτωση και την προετοιμασία των δύο *datasets*: το *HASYv2* για τα μαθηματικά σύμβολα και το *MNIST* για τα ψηφία.

```
ds = deeplake.load("hub://activeloop/hasy-test")

ADDITION = 113
SUBTRACTION = 112
MULTIPLICATION = 131
DIVISION = 138

addition_images = []
subtraction_images = []
multiplication_images = []
division_images = []

def resizeImage(image):
    # Resize to 28x28
    resized_image = cv2.resize(image, (28, 28), interpolation=cv2.INTER_AREA)
```

```

# Convert to grayscale
grayscale_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
# Normalize to [0, 1] and add a channel dimension
normalized_image = grayscale_image / 255.0
# Invert image
inverted_image = 1 - normalized_image
return inverted_image.reshape(28, 28, 1)

# Iterate through the dataset using enumerate for more efficiency
for i, sample in enumerate(ds):
    label = sample['label'].numpy()

    if label in [ADDITION, SUBTRACTION, MULTIPLICATION, DIVISION]:
        image = resizeImage(sample['images'].numpy())
        if label == ADDITION:
            addition_images.append(image)
        elif label == SUBTRACTION:
            subtraction_images.append(image)
        elif label == MULTIPLICATION:
            multiplication_images.append(image)
        elif label == DIVISION:
            division_images.append(image)

# Step 1: Load and Normalize MNIST Dataset
(train_images, train_labels), (_, _) = mnist.load_data()
train_images = train_images / 255.0 # Normalize pixel values to [0, 1]

# Function to Retrieve a Random MNIST Image for a Given Digit
def get_mnist_image(digit):
    indices = np.where(train_labels == digit)[0]
    selected_index = random.choice(indices)
    return train_images[selected_index]

```

## 2. Γεννήτρια Δεδομένων Εκπαίδευσης

Αυτό το τμήμα δημιουργεί το dataset εισόδου με εικόνες από το *MNIST* και το *HASYv2* και υπολογίζει τα αποτελέσματα των αριθμητικών πράξεων. Δημιουργεί ένα ισορροπημένο *dataset* με βάση τα 4 μαθηματικά σύμβολα και επιστρέφει εικόνες εισόδου και του αντίστοιχου *operator label*.

```

def generate_dataset_with_operator_images(num_samples=50000, operator_images=None):
    input1_images, input2_images = [], []
    operator_images, output1_images, output2_images, operators = [], [], [], []
    # Map operators to integers
    operator_map = {'+': 0, '-': 1, '*': 2, '/': 3}
    # Define target samples per operator
    operator_samples = {'+': 12500, '-': 12500, '*': 12500, '/': 12500}
    # Counters for each operator
    operator_counts = {'+': 0, '-': 0, '*': 0, '/': 0}

    for _ in range(num_samples):
        # Randomly select two digits and an operator
        digit1 = random.randint(0, 9)
        digit2 = random.randint(0, 9)

        # Generate dataset with balanced operators
        operator = random.choice(['+', '-', '*', '/'])
        if(operator_counts['+'] < operator_samples['+']):
            operator = '+'
            operator_counts[operator] += 1
        elif(operator_counts['-'] < operator_samples['-']):
            operator = '-'
            operator_counts[operator] += 1
        elif(operator_counts['*'] < operator_samples['*']):
            operator = '*'
            operator_counts[operator] += 1
        elif(operator_counts['/'] < operator_samples['/']):
            operator = '/'
            operator_counts[operator] += 1

```

```

# Compute the result of the operation
if operator == '+':
    total = digit1 + digit2
elif operator == '-':
    total = digit1 - digit2
elif operator == '*':
    total = digit1 * digit2
elif operator == '/':
    total = digit1 // digit2 if digit2 != 0 else 0 # Avoid division by zero

# Split the result into two digits
total_str = f"{abs(total):02}" # Ensure two digits
output1 = int(total_str[0]) # First digit
output2 = int(total_str[1]) # Second digit

# Retrieve MNIST images
input1 = get_mnist_image(digit1)
input2 = get_mnist_image(digit2)
operator_input = get_op_image(operator)
output1_img = get_mnist_image(output1)
output2_img = get_mnist_image(output2)

# Append to dataset
input1_images.append(input1)
input2_images.append(input2)
operator_images.append(operator_input) # Append operator image
output1_images.append(output1_img)
output2_images.append(output2_img)
operators.append(operator_map[operator])

# Convert to NumPy arrays and reshape for Keras compatibility
return (
    np.array(input1_images).reshape(-1, 28, 28, 1),
    np.array(input2_images).reshape(-1, 28, 28, 1),
    np.array(operator_images).reshape(-1, 28, 28, 1),
    np.array(output1_images).reshape(-1, 28, 28, 1),
    np.array(output2_images).reshape(-1, 28, 28, 1),
    np.array(operators)
)

```

### 3. Δομή και Δημιουργία του Μοντέλου

Αυτό το τμήμα κατασκευάζει τη δομή του νευρωνικού δικτύου: encoder, decoder, και layer για την πρόβλεψη του συμβόλου. Ρόλος του είναι η δημιουργία κοινών *CNN* για τις τρεις εισόδους, η δημιουργία του *latent space* και των αποκωδικοποιητών για τις δύο εξόδους εικόνας και η πρόβλεψη του *operator* μέσω του *dense layer*.

```

def build_shared_cnn(input_shape, conv_layers, filters, kernel_size, pool_size, activation):
    input_layer = Input(shape=input_shape)
    x = input_layer
    for i in range(conv_layers):
        x = Conv2D(filters[i], (kernel_size[i], kernel_size[i]), activation=activation,
padding='same')(x)
        x = MaxPooling2D((pool_size[i], pool_size[i]))(x)
    x = Flatten()(x)
    return input_layer, x

# Parameterized decoder
def build_decoder(latent, dense_size, output_shape, transposed_conv_layers, filters, kernel_size,
strides, activation, name):
    x = Dense(dense_size, activation=activation)(latent)
    x = Reshape(output_shape)(x)
    for i in range(transposed_conv_layers):
        x = Conv2DTranspose(filters[i], (kernel_size[i], kernel_size[i]), strides=(strides[i],
strides[i]), activation=activation, padding='same')(x)

# Build shared CNNs for inputs
input1, features1 = build_shared_cnn(input_shape, conv_layers, filters, kernel_size, pool_size,
activation)
input2, features2 = build_shared_cnn(input_shape, conv_layers, filters, kernel_size, pool_size,
activation)
operator_input, operator_features = build_shared_cnn(input_shape, conv_layers, filters, kernel_size,
pool_size, activation)

```

```

# Merge features
merged_features = concatenate([features1, operator_features, features2])

# Latent space
latent = Dense(latent_size, activation=activation)(merged_features)
# Decoders for two outputs
output1 = build_decoder(latent, dense_size, output_shape, transposed_conv_layers,
transposed_filters, transposed_kernel_size, transposed_strides, activation, 'output1')
output2 = build_decoder(latent, dense_size, output_shape, transposed_conv_layers,
transposed_filters, transposed_kernel_size, transposed_strides, activation, 'output2')

# New output for operator classification
operator_output = Dense(4, activation='softmax', name='operator_output')(latent)

# Define the model
model = Model(inputs=[input1, input2, operator_input], outputs=[output1, output2, operator_output])

```

#### 4. Εκπαίδευση και Αποθήκευση του Μοντέλου

Εκπαίδευση του μοντέλου με τις τρεις εξόδους, χρήση διαφορετικών *loss functions* και *loss weights* για τις εξόδους.

```

model.compile(
optimizer=Adam(learning_rate=learning_rate),
loss={
    'output1': 'mse',                                # Loss for Image Output 1
    'output2': 'mse',                                # Loss for Image Output 2
    'operator_output': 'categorical_crossentropy',    # Loss for Operator
},
loss_weights={
    'output1': 1.0,                                  # Weight for Image Output 1
    'output2': 1.0,                                  # Weight for Image Output 2
    'operator_output': 2.0,                            # Weight for Operator
},
metrics={
    'output1': ['mse'],
    'output2': ['mse'],
    'operator_output': ['accuracy', 'categorical_crossentropy']
}

# Train the model
history = model.fit(
    [X1, X2, Ops],                                     # Input data: two digit images and one operator image
    [Y1, Y2, Op_labels_categorical],                  # Target data: two output images
    batch_size=batch_size,                            # Number of samples per batch
    epochs=epochs,                                    # Number of training epochs
    validation_split=0.2,                            # Use 20% of data for validation
    verbose=1,                                       # Print training progress
    shuffle = True
)

model.save(model_name, save_format='tf')

```

Το αρχείο *MNIST\_Image\_Identifier\_Training.py* εκπαιδεύει το δίκτυο κατηγοριοποίησης εικόνων *MNIST*.

#### 1. Ορισμός και Αρχιτεκτονική Μοντέλου

Ορίζεται η βασική αρχιτεκτονική του νευρωνικού δικτύου με συνελικτικά επίπεδα, επίπεδα *Pooling*, πλήρως συνδεδεμένο επίπεδο, *Dropout* για *regularization*, και επίπεδο εξόδου για ταξινόμηση 10 κατηγοριών.

```

model = Sequential([
    # Convolutional Layer 1
    Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)),

```

```

# Convolutional Layer 2
Conv2D(64, (3, 3), activation='relu', padding='same'),
MaxPooling2D(pool_size=(2, 2)),

# Flatten and Fully Connected Layers
Flatten(),
Dense(128, activation='relu'),

# Dropout for regularization
Dropout(0.5),

# Output Layer
Dense(10, activation='softmax')
])

```

## 2. Συμπλήρωση και Επισκόπηση του Μοντέλου

Καθορίζονται ο βελτιστοποιητής (*Adam*), η συνάρτηση απώλειας (*categorical crossentropy*) και η ακρίβεια ως μέτρο αξιολόγησης. Η σύνοψη του μοντέλου (*summary*) παρουσιάζει την αρχιτεκτονική και τον αριθμό παραμέτρων ανά επίπεδο.

```

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Display model summary
model.summary()

```

## 3. Φόρτωση και Προεπεξεργασία Δεδομένων *MNIST*

Γίνεται φόρτωση του συνόλου δεδομένων *MNIST*, κανονικοποίηση των τιμών των *pixels* στο [0,1] και μετατροπή των ετικετών σε μορφή *one – hot encoding* για πολυκατηγορική ταξινόμηση.

```

(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
X_test = X_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

```

## 4. Εκπαίδευση και Αποθήκευση του Μοντέλου

Το μοντέλο εκπαιδεύεται και αποθηκεύεται σε αρχείο *HDF5* για μελλοντική χρήση.

```

history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=32)
model.save(model_name, save_format='tf')

```

Στο αρχείο *Multiple\_Operations\_Tester.py* ελέγχονται οι έξοδοι του μοντέλου.

### 1. Φόρτωση Εκπαιδευμένου Μοντέλου και Ταξινομητή Ψηφίων

Φορτώνονται τα προεκπαιδευμένα μοντέλα: το κύριο μοντέλο για την πρόβλεψη αποτελεσμάτων και ο ταξινομητής ψηφίων για αναγνώριση των προβλεπόμενων εξόδων.

```
# Load the model from the .h5 file
model = load_model('ModelResults/final_50k_12iter.h5', custom_objects=custom_objects)

# Load pretrained MNIST classifier (replace with your actual model)
digit_classifier = load_model('ModelResults/identifier.h5')
```

## 2. Φόρτωση και Προεπεξεργασία Συνόλου Δεδομένων *MNIST* και *HASYv2*

Γίνεται φόρτωση και προεπεξεργασία των δεδομένων *MNIST* για τα ψηφία και *HASYv2* για τους τελεστές, ώστε να είναι έτοιμα για εισαγωγή στο δίκτυο.

```
# Load MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Load and preprocess symbol dataset
ds = deeplake.load("hub://activeloop/hasy-test")
```

## 3. Προβολή Εισόδων και Προβλεπόμενων Εξόδων

Η συνάρτηση *display\_prediction* προβάλλει τις εικόνες εισόδου, τις προβλεπόμενες εξόδους και τις κορυφαίες προβλέψεις του ταξινομητή.

```
def display_prediction(input1, input2, operator_input, output1, output2, top_guesses_output1,
top_guesses_output2):
    size = (28, 28)
    output_size = (28, 28)

    plt.figure(figsize=(12, 5))

    # Input 1
    plt.subplot(1, 7, 1)
    plt.imshow(input1.reshape(size), cmap='gray')
    plt.title("Input 1")
    plt.axis('off')

    # Op
    plt.subplot(1, 7, 2)
    plt.imshow(operator_input.reshape(size), cmap='gray')
    plt.title("Input 2")
    plt.axis('off')

    # Input 2
    plt.subplot(1, 7, 3)
    plt.imshow(input2.reshape(size), cmap='gray')
    plt.title("Input 3")
    plt.axis('off')

    # Predicted Output 1
    plt.subplot(1, 7, 4)
    plt.imshow(output1.reshape(output_size), cmap='gray')
    plt.title("Predicted Output 1")
    plt.axis('off')

    # Predicted Output 2
    plt.subplot(1, 7, 5)
    plt.imshow(output2.reshape(output_size), cmap='gray')
    plt.title("Predicted Output 2")
    plt.axis('off')

    # Top guesses for Output 1
    plt.subplot(1, 7, 6)
    plt.text(0.5, 0.55, f"Top 1: {top_guesses_output1[0]}", fontsize=12, ha='center')
    plt.text(0.5, 0.45, f"Top 2: {top_guesses_output1[1]}", fontsize=12, ha='center')
    plt.axis('off')

    # Top guesses for Output 2
    plt.subplot(1, 7, 7)
    plt.text(0.5, 0.55, f"Top 1: {top_guesses_output2[0]}", fontsize=12, ha='center')
    plt.text(0.5, 0.45, f"Top 2: {top_guesses_output2[1]}", fontsize=12, ha='center')
    plt.axis('off')

    plt.show()
```

#### 4. Δοκιμή του Μοντέλου με Τυχαίες Εικόνες

Γίνεται δοκιμή του μοντέλου με τυχαία δείγματα, προβλέπονται τα αποτελέσματα και εμφανίζονται οι προβλέψεις μαζί με τις κορυφαίες πιθανότητες για τις εξόδους.

```
for i in range(0,30):
    operator = random.choice(['+', '-', '*', '/'])
    match (operator):
        case '+': image = random.choice(addition_images)
        case '-': image = random.choice(subtraction_images)
        case '*': image = random.choice(multiplication_images)
        case '/': image = random.choice(division_images)

    image1 = train_images[np.random.randint(0, 500)]
    image2 = train_images[np.random.randint(0, 500)]

    # Assume test_input1 and test_input2 are 2D tensors (28, 28)
    test_input1 = np.expand_dims(image1, axis=0) # Add batch dimension
    test_input2 = np.expand_dims(image2, axis=0) # Add batch dimension
    operator_image = np.expand_dims(image, axis=0)

    # Predict outputs
    predicted_output1, predicted_output2, predicted_operator = model.predict(
        [test_input1, test_input2, operator_image]
    )

    # Get top 2 guesses for output images
    top_guesses_output1 = np.argsort(-digit_classifier.predict(predicted_output1)[0])[:2]
    top_guesses_output2 = np.argsort(-digit_classifier.predict(predicted_output2)[0])[:2]

    # Map the predicted operator index to its corresponding symbol
    operator_mapping = {0: '+', 1: '-', 2: '*', 3: '/'}
    predicted_operator_symbol = operator_mapping[np.argmax(predicted_operator)]
```

## Πηγές

### NeuPSL: Neural Probabilistic Soft Logic

Connor Pryor, Charles Dickens, Eriq Augustine, Alon Albalak, William Wang, Lise Getoor  
arXiv preprint arXiv:2205.14268, 2022.

Το πλήρες έγγραφο μπορεί να βρεθεί στον παρακάτω σύνδεσμο:

<https://arxiv.org/abs/2205.14268>