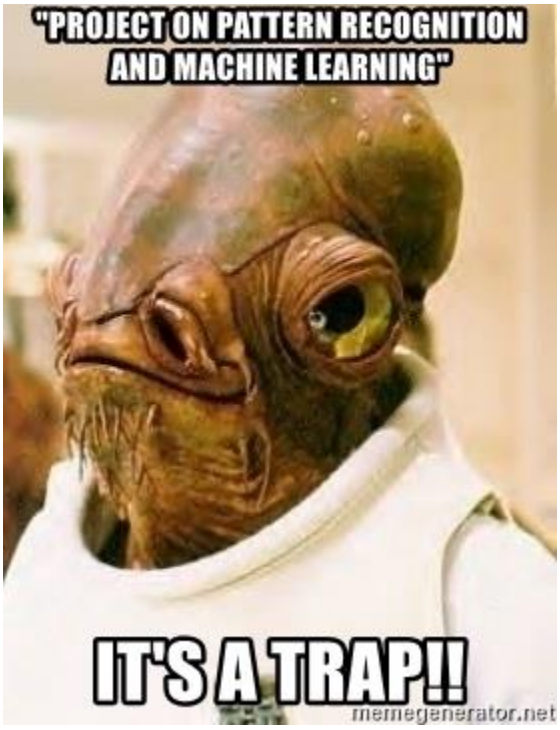
ΑΝΑΓΝΩΡΙΣΗ ΠΡΟΤΥΠΩΝ 2020

Ατομική εξαμηνιαία εργασία



Υπεύθυνος καθηγητής :
Νικόλαος Μητιανούδης

Σάββας Λιάπης 57403

Υπεύθυνοι εργαστηρίου :
Γεώργιος-Αλέξης Ιωαννάκης
Ιωάννης Βασιλάκης

1. Απλό Τεχνητό Νευρωνικό Δίκτυο/Artificial Neural Network (ANN)

Ερώτημα 1.a.

Υλοποιήστε ένα απλό τεχνητό νευρωνικό δίκτυο (fully connected) με δύο κρυφά επίπεδα 256 και 128 νευρώνων αντίστοιχα. Προσαρμόστε το επίπεδο εισόδου (input layer)και το επίπεδο εξόδου (output layer) κατάλληλα για το dataset που πρέπει να χρησιμοποιήσετε.

Απάντηση 1.a.

```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.utils import np_utils
from sklearn.datasets import make_classification
```

Πριν υλοποιήσουμε το νευρωνικό δίκτυο του ερωτήματος α και αφού εισάγου με το dataset είναι αναγκαίο να εισάγουμε κάποιες βιβλιοθήκες όπως φαίνονται παραπάνω.

- Οι λειτουργίες Wrapper μπορούν να χρησιμοποιηθούν για να διευκολύνουν τη σύνταξη προγραμμάτων υπολογιστή, είναι χρήσιμες για την ανάπτυξη εφαρμογών που χρησιμοποιούν συναρτήσεις βιβλιοθήκης τρίτων. Ένας wrapper μπορεί να γραφτεί για καθεμία από τις λειτουργίες τρίτων και να χρησιμοποιηθεί στην εγγενή εφαρμογή. Σε περίπτωση αλλαγής ή ενημέρωσης των λειτουργιών τρίτων, μόνο τα περιτυλίγματα στην εγγενή εφαρμογή πρέπει να τροποποιηθούν σε αντίθεση με την αλλαγή όλων των παρουσιών συναρτήσεων τρίτων στην εγγενή εφαρμογή. Στην ουσία είναι ένα εργαλείο για την ευκολότερη διαχείριση των συναρτήσεων που θα χρησιμοποιήσουμε . Στην δική μας περίπτωση εισάγουμε τον Keras Classifier. Έπειτα από το keras εισάγουμε ότι άλλο θα χρειαστούμε
- Τα υπόλοιπα έχουν να κάνουν με την αρχιτεκτονική του δικτύου και αναλύονται παρακάτω

```
#creating object for exercise 1a
model = Sequential()

# 1st hidden layer
model.add(Dense(256,input_shape=(4096,)))
model.add(Dropout(0.2))

# 2nd hidden layer
model.add(Dense(128))
model.add(Dropout(0.2))

#output neuron
model.add(Dense(10))
#softmax needed in order to get as propability
model.add(Activation('softmax'))

model.summary()
```

Ορίζουμε model = Sequential () το οποίο μας επιτρέπει να «χτίσουμε» το μοντέλο μας επίπεδο προς επίπεδο, σειριακά. Στην συνέχεια υλοποιούμε το μοντέλο μας (επίπεδο προς επίπεδο) :

1. Στο επίπεδο εισόδου θα δώσουμε ένα διάνυσμα εισόδου μεγέθους 4096 . Το νούμερο αυτό προκύπτει ως εξής : το σύνολο δειγμάτων μας είναι ένα σύνολο δυσδιάστατων εικόνων , οπότε πριν το εισάγουμε μετασχηματίζουμε τις 2 διαστάσεις της εικόνας σε 1 (ως γινόμενο των 2) . Οι εικόνες του dataset μας όπως έχουν εισαχθεί έχουν διάσταση 64x64 . Το γινόμενο 64x64 δίνει 4096.
2. Το πρώτο κρυφό επίπεδο θα έχει 256 νευρώνες (όπως ορίζεται από την άσκηση) και αυτό το δηλώνουμε με την εντολή model.add(Dense(αριθμός νευρώνων)). Με την εντολή Dence ορίζουμε ότι όλες οι έξοδοι του προηγούμενου επιπέδου θα συνδέονται με όλες τις εισόδους του παρόντος δηλαδή έχουμε fully connected layer.
3. Στην συνέχεια δηλώνουμε το Dropout με την εντολή :
   model.add(Dropout(ποσοστό νευρώνων που απενεργοποιούνται))
   Το dropout είναι μια τεχνική που χρησιμοποιείται για να αποτρέψει το overfit ενός μοντέλου. Το Dropout λειτουργεί ρυθμίζοντας τυχαία τα εξερχόμενα άκρα των hidden layers σε 0 σε κάθε ενημέρωση της προπόνησης.
4. Στην συνέχεια ορίζουμε το δεύτερο κρυφό επίπεδο θέτοντας των αριθμό των νευρώνων 128. Παρατηρούμε ότι πλέον δεν υπάρχει η εντολή : ,input_shape=(4096,) καθώς πλέον τα δεδομένα μας έχουν εισαχθεί και οι νευρώνες παίρνουν σαν είσοδο τις εξόδους του προηγούμενου κρυφού επιπέδου.

5. Στην συνέχεια ορίζουμε το επίπεδο εξόδου ορίζοντας ως αριθμό νευρώνων των αριθμό των κλάσεων που κάνουμε classification στην δικιά μας περίπτωση έχουμε 10 κλάσεις
6. Στο επίπεδο εξόδου εισάγουμε και την συνάρτηση ενεργοποίησης softmax η οποία κανονικοποιεί την έξοδο σε μία κατανομή πιθανότητας σε σχέση με τις προβλεπόμενες κλάσεις.

Για να δούμε την τελική δομή και το σύνολο των παραμέτρων χρησιμοποιούμε την εντολή model.summary() και παίρνουμε το εξής αποτέλεσμα

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               1048832
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
_________________________________________________________________
activation (Activation)      (None, 10)                0
=================================================================
Total params: 1,083,018
Trainable params: 1,083,018
Non-trainable params: 0
```

Παρατηρούμε ότι :

Στο πρώτο κρυφό επίπεδο οι παράμετροι που περιμένουμε να έχουμε είναι :
αριθμός των νευρώνων *διάνυσμα εισόδων + biases (που είναι όσα ο αριθμός νευρώνων) .
Αρά 256*4096+256=1048832
σε όλα τα επίπεδα εκτός από τα dence έχουμε 0 παραμέτρους που μαθαίνονται

Ερώτημα 1.b.

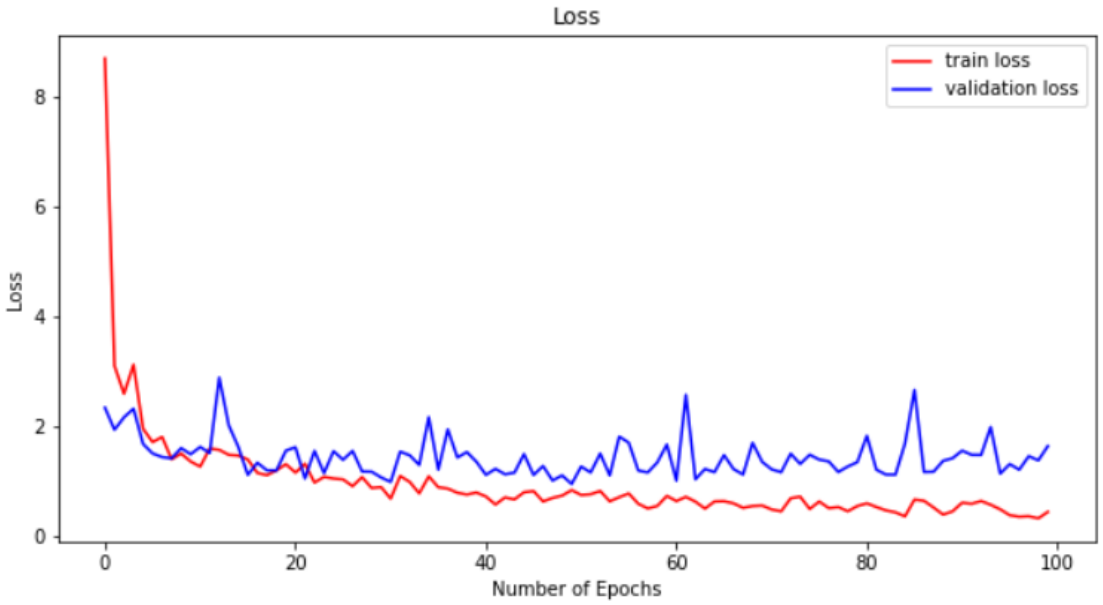
Παραγωγή των γραφημάτων train/test – loss ανά εποχή. Εμφανίζεται το πρόβλημα underfitting/overfitting; Αν ναι, να προτείνετε έναν τρόπο διαχείρισης του φαινομένου. Επίσης να αιτιολογήσετε ποιος είναι ο βέλτιστος αριθμός εποχών εκπαίδευσης. Για τον βέλτιστο αριθμό εποχών εκπαίδευσης, ζητείται να εξαχθούν οι μετρικές: accuracy, precision recall, f1 score, confusion matrix.

Απάντηση 1.b.

Καθώς εκπαιδεύουμε ένα δίκτυο παρατηρούμε ότι τα validation και training σφάλματα μειώνονται. Ωστόσο πολλές φορές όταν παρατηρείται η αύξηση του validation σφάλματος τότε λέμε ότι έχουμε overfitting . Αυτό συμβαίνει γιατί το δίκτυο υπερ-εκπαιδεύεται στα training δεδομένα του χάνοντας την δυνατότητα να γενικεύει. Όταν συναντάται αυτό το πρόβλημα κάποιες μέθοδοι αντιμετώπισης του είναι :

- Μείωση της πολυπλοκότητας του μοντέλου. Για να μειώσουμε την πολυπλοκότητα, μπορούμε απλά να αφαιρέσουμε στρώματα ή να μειώσουμε τον αριθμό των νευρώνων για να κάνουμε το δίκτυο μικρότερο.
- Early stopping, δηλαδή διακοπή της εκπαίδευσης όταν παρατηρείται η αύξηση του validation σφάλματος
- Εφαρμογή Data augmentation , δηλαδή αύξηση των δειγμάτων του dataset και επεξεργασία των εικόνων για να επιτευχθεί καλύτερη γενίκευση κατά την εκμάθηση
- Χρήση dropouts . Η τεχνική αυτή κάνει drop σε random νευρώνες κατά την εκπαίδευση σε κάθε εποχή. Όταν απενεργοποιούμε διαφορετικούς νευρώνες είναι σαν να κάνουμε εκπαίδευση με διαφορετικά δίκτυα, και αυτό βελτιώνει το overfitting ως έναν βαθμό.

Στο δικό μας δίκτυο παρατηρούμε ότι πετυχαίνουμε το καλύτερο validation accuracy στην εποχή 49 . από εκεί και μετά το validation accuracy δεν βελτιώνεται ενώ το train accuracy συνεχίζει να βελτιώνεται . Αυτό σημαίνει ότι εμφανίζεται το πρόβλημα του overfitting



Για την καλύτερη εποχή οι μετρικές που παίρνουμε είναι :

```
best epoch:  49
best epoch validation accuracy:  0.7409201264381409
best epoch train accuracy:  0.7311320900917053
best epoch precision:  0.7760277390480042
best epoch recall:  0.7034574747085571
best epoch f1:  0.7373949289321899
```

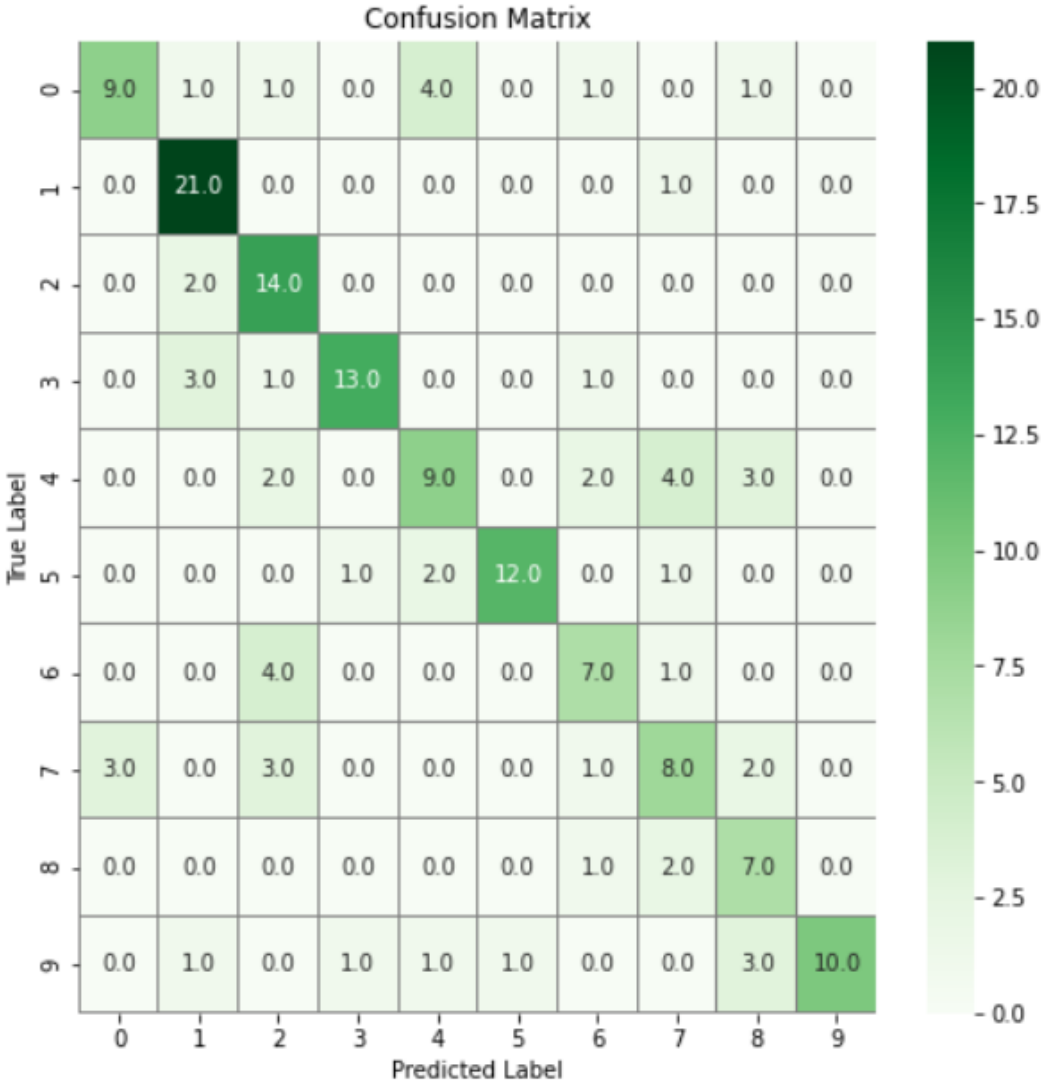
Accuracy - Η ακρίβεια είναι ένα καλό μέτρο απόδοσης και είναι απλώς μια αναλογία σωστά προβλεπόμενης παρατήρησης προς τις συνολικές παρατηρήσεις. Η ακρίβεια είναι ένα εξαιρετικό μέτρο, αλλά μόνο όταν έχουμε συμμετρικά σύνολα δεδομένων όπου οι τιμές ψευδώς θετικών και ψευδών αρνητικών είναι σχεδόν ίδιες. Επομένως, πρέπει να εξετάσουμε άλλες παραμέτρους για να αξιολογήσετε την απόδοση του μοντέλου μας.

Precision - Η ακρίβεια είναι ο λόγος των σωστά προβλεπόμενων θετικών παρατηρήσεων προς τις συνολικές προβλεπόμενες θετικές παρατηρήσεις.

Recall (Ευαισθησία) - Ανάκληση είναι ο λόγος των σωστά προβλεπόμενων θετικών παρατηρήσεων προς όλες τις παρατηρήσεις στην πραγματική τάξη - ναι.

F1 - Το σκορ F1 είναι ο σταθμισμένος μέσος όρος του accuracy και recall. Επομένως, αυτό το σκορ λαμβάνει υπόψη τόσο ψευδώς θετικά όσο και ψευδώς αρνητικά. Δεν είναι τόσο εύκολο να κατανοηθεί όσο η ακρίβεια, αλλά το F1 είναι συνήθως πιο χρήσιμο από την ακρίβεια, ειδικά αν έχετε μια άνιση κατανομή τάξης.

Το confusion matrix είναι ένας πρακτικός τρόπος να δούμε πόσα δεδομένα έχουν ταξινομηθεί σωστά και πόσα λανθασμένα και να εμβαθύνουμε παρατηρώντας ποια δεδομένα συγχέονται ευκολότερα για να κάνουμε περαιτέρω προσαρμογές στο δίκτυό μας που θα τονίζονται χαρακτηριστικά στα οποία τα δεδομένα που συγχέονται εύκολα , είναι διαφορετικά .



Στο δικό μας δίκτυο βλέπουμε ότι τα δεδομένα της κλάσης 0 αναγνωρίζονται σαν δεδομένα κλάσης 4, ακόμη πιο έντονα δεδομένα κλάσης 4 συγχέονται με 6,7,8 και καταλαβαίνουμε πως προκύπτει από το dataset μας αυτό, καθώς στην νοηματική τα νούμερα 4,6,7,8 μοιάζουν πολύ μεταξύ τους.

Ερώτημα 1.c.

Δοκιμάστε να κανονικοποιήσετε τις εικόνες στο διάστημα [-1,1]. Ταυτόχρονα, προσθέστε την συνάρτηση ενεργοποίησης (activation function) ReLU (Rectified Linear Unit) ως έξοδο κάθε νευρώνα στα κρυφά επίπεδα (hidden layer). Υπάρχει αλλαγή στην ακρίβεια (accuracy); Ο αριθμός των εποχών πριν συμβεί το overfit έχει μεταβληθεί; Γιατί; Ποια είναι η μεταβολή στον αριθμό των παραμέτρων του δικτύου;

Απάντηση 1.c.

για την κανονικοποίηση στο διάστημα [-1,1] χρησιμοποιούμε την συνάρτηση MinMaxScaler από την sklearn.preprocessing

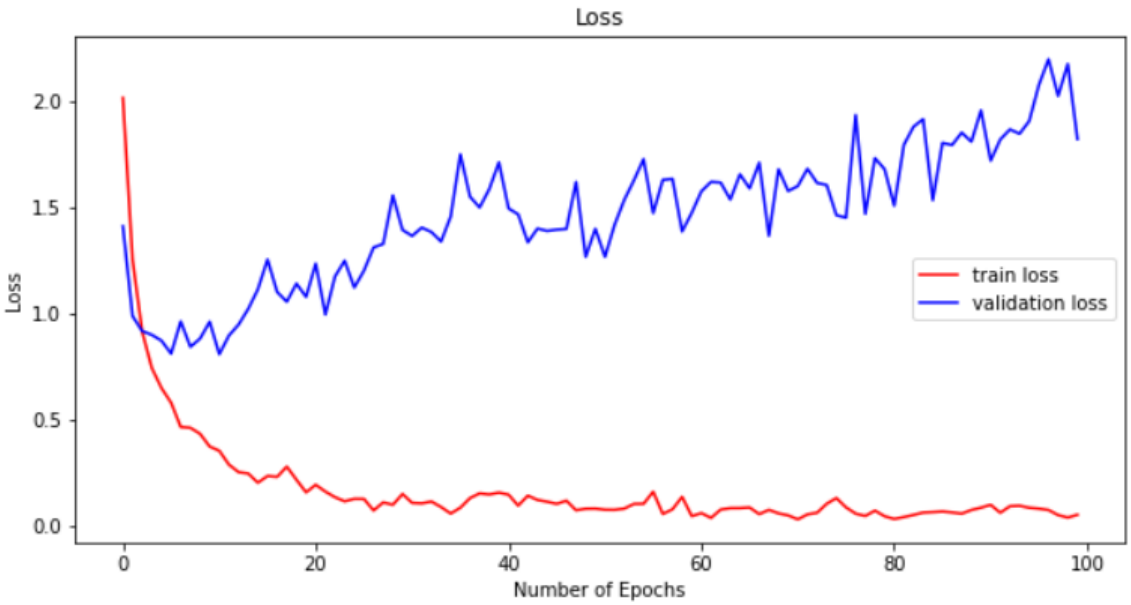
```
scaler = MinMaxScaler(feature_range=(-1, 1))
x_train = scaler.fit_transform(x_train)
x_val = scaler.fit_transform(x_val)
x_test = scaler.fit_transform(x_test)
```

```
#creating object for exercise 1c
model1c = Sequential()

# 1st hidden layer
model1c.add(Dense(256,input_shape=(4096,)))
model1c.add(Activation('relu'))
model1c.add(Dropout(0.2))

# 2nd hidden layer
model1c.add(Dense(128))
model1c.add(Activation('relu'))
model1c.add(Dropout(0.2))

#output neuron
model1c.add(Dense(10))
#softmax needed in order to get as propability
model1c.add(Activation('softmax'))

model1c.summary()
```

Η αρχιτεκτονική του δικτύου αφού προσθέσουμε και την Relu θα είναι αυτή που φαίνεται αριστερά

Οι παράμετροι του δικτύου βλέπουμε ότι δεν έχουν αυξομειωθεί και αυτό άλλωστε περιμέναμε να γίνει.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 256)               1048832
_________________________________________________________________
activation_2 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               32896
_________________________________________________________________
activation_3 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1290
_________________________________________________________________
activation_4 (Activation)    (None, 10)                0
=================================================================
Total params: 1,083,018
Trainable params: 1,083,018
Non-trainable params: 0
```




```
best epoch:  10
best epoch validation accuracy:  0.7530266046524048
best epoch train accuracy:  0.8773584961891174
```

Παρατηρούμε ότι πετυχαίνουμε καλύτερο validation και train accuracy και πολύ γρηγορότερα από πριν (η καλύτερη εποχή τώρα είναι η 10η σε σύγκριση με πρίν που ήταν η 49η) . Αυτό μπορεί να οφείλεται στο ότι οι αρνητικές τιμές (που είναι οι τιμές χαμηλότερης φωτεινότητας και δεν μας προσφέρουν κάποια σημαντική πληροφορία) γίνονται 0

Ερώτημα 1.d.

Δοκιμάστε παράλληλα με τις προσθήκες του προηγούμενου ερωτήματος να εφαρμόσετε και batch normalization σε κάθε επίπεδο. Ποια είναι η μεταβολή στο μέγεθος του δικτύου; Αναμένεται να υπάρχει και μεταβολή του χρόνου επεξεργασίας του δικτύου; Τι επιτυγχάνει ο αλγόριθμος batch normalization;

Απάντηση 1.d.

```
#creating object for exercise 1d
model1d = Sequential()

# 1st hidden layer
model1d.add(Dense(256,input_shape=(4096,)))
model1d.add(BatchNormalization())
model1d.add(Activation('relu'))
model1d.add(Dropout(0.2))

# 2nd hidden layer
model1d.add(Dense(128))
model1d.add(BatchNormalization())
model1d.add(Activation('relu'))
model1d.add(Dropout(0.2))

#output neuron
model1d.add(Dense(10))
#softmax needed in order to get as propability
model1d.add(Activation('softmax'))

model1d.summary()
```

Για να εισάγουμε το batch normalization επίπεδο αφού κάνουμε την εισαγωγή :

from keras.layers import BatchNormalization

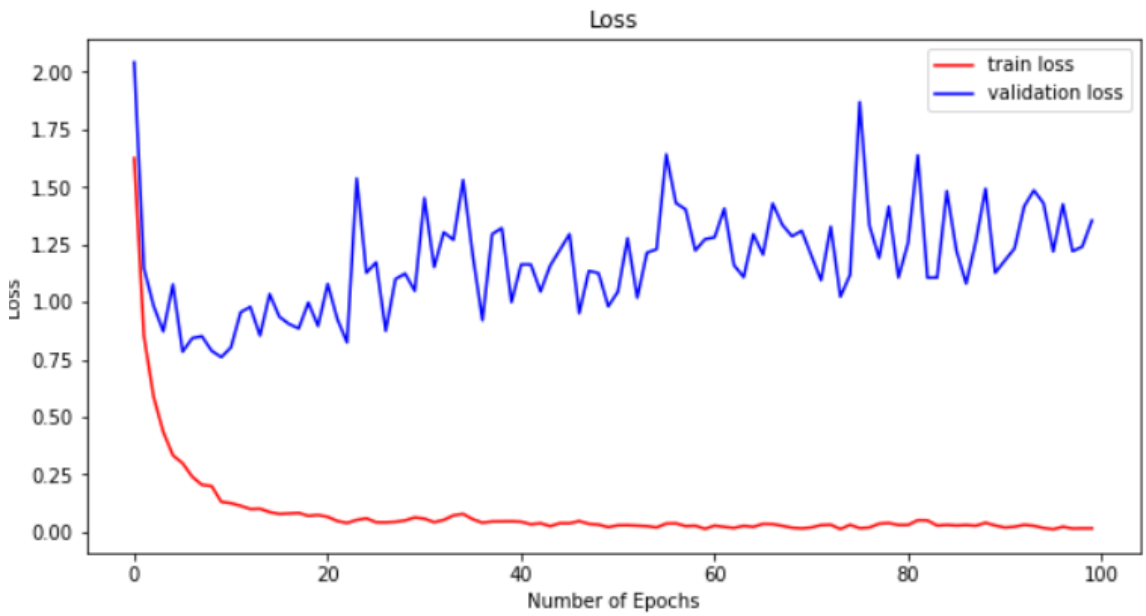
το εισάγουμε με το ίδιο τρόπου που εισάγουμε και τα άλλα επίπεδα .

Ο αλγόριθμος batch normalization μπορεί να επιφέρει γρηγορότερη εκπαίδευση του δικτύου. Παρόλο που η κάθε εποχή μπορεί να χρειαστεί παραπάνω χρόνο από ότι πριν, συνολικά η εκπαίδευση θα συγκλίνει πολύ πιο γρήγορα. Πάνω σε αυτό καταλαβαίνουμε ότι επιτρέπει και υψηλότερα learning rates . Επίσης ο αλγόριθμος ρυθμίζει τι τιμές (προσθέτει κάποιο είδος θορύβου). Αυτό μπορεί να επηρεάσει και την επίδοσή των συναρτήσεων ενεργοποίησης καθώς οι τιμές που εισέρχονται στις activation function (για αυτό προτιμάται να εφαρμόζεται πριν την συνάρτηση ενεργοποίησης) έχουν υποστεί κάποιου είδους κανονικοποίηση και ως εκ τούτου πολλές συναρτήσεις ενεργοποίησής τείνουν να λειτουργούν αποτελεσματικά σε περιπτώσεις που δεν θα συνέβαινε αυτό.

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 256)	1048832
batch_normalization (BatchNo	(None, 256)	1024
activation_5 (Activation)	(None, 256)	0
dropout_6 (Dropout)	(None, 256)	0
dense_10 (Dense)	(None, 128)	32896
batch_normalization_1 (Batch	(None, 128)	512
activation_6 (Activation)	(None, 128)	0
dropout_7 (Dropout)	(None, 128)	0
dense_11 (Dense)	(None, 10)	1290
activation_7 (Activation)	(None, 10)	0

Total params: 1,084,554
Trainable params: 1,083,786
Non-trainable params: 768

όσον αφορά τις παραμέτρους βλέπουμε ότι με την εισαγωγή του batch normalization έχουμε την είσοδο 768 παραμέτρων ακόμα και οι οποίες όμως δεν επιδέχονται μαθήσεως.




```
best epoch:  9
best epoch validation accuracy:  0.7433413863182068
best epoch train accuracy:  0.9737196564674377
```

Βλέπουμε ότι η καλύτερη εποχή βελτιώνεται για λίγο αλλά το validation accuracy είναι ελάχιστα μικρότερο ενώ το train accuracy είναι πολύ υψηλότερο (οπότε η εκπαιδευση συγκλίνει πολύ γρηγορότερα) . Επίσης αν συγκρίνουμε τα test accuracies του 1.c (αριστερά) και 1.d (δεξιά)

```
Test score:  3.7405569553375244
Test accuracy:  0.6848484873771667
```

```
Test score:  2.368924140930176
Test accuracy:  0.6484848260879517
```

Βλέπουμε ότι το testing είναι καλύτερο χωρίς το batch normalization (αλλά αυτό μπορεί να φταίει στο ότι η εκπαίδευση δεν σταμάτησε στην καλύτερη εποχή)

Ερώτημα 1.e.

Υλοποιήστε διάφορες δομές τεχνητών νευρωνικών δικτύων και εντοπίστε αυτό που επιλύει το πρόβλημα της ταξινόμησης με μεγαλύτερη ακρίβεια. ΣΗΜΕΙΩΣΗ: Μπορείτε για παράδειγμα να πειραματιστείτε με τον αριθμό των επιπέδων, των νευρώνων ανά επίπεδο, τις συναρτήσεις ενεργοποίησης, τον αλγόριθμο βελτιστοποίησης (optimization algorithm), την προ-επεξεργασία των εικόνων του dataset, μείωση διαστάσεων (dimensionality reduction). Είστε ελεύθερες/ελεύθεροι να υλοποιήσετε ότι θεωρείτε ότι θα βοηθήσει στην αύξηση της ακρίβειας.

Απάντηση 1.e.

Παραλλαγή	Αριθμός παραμέτρων	Καλύτερη εποχή	Validation accuracy	Train accuracy	Test accuracy
(1e1): 3 hidden (256,128,64)	1,092,426	10	0.7699	0.9366	0.7272
(1e2): 4 hidden (256,128,64,32)	1,094,314	34	0.8014	0.9555	0.7090
(1e3): 4 hidden (128,128,64,64)	555,530	17	0.7845	0.9130	0.6666
(1e4): 5 hidden (256,128,128,64,32)	1,111,338	11	0.7384	0.8497	0.6606
(1e5): (1e2)+ activation : selu	1,094,314	6	0.6973	0.8402	0.5939
(1e6): (1e2) + activation : elu	1,094,314	18	0.7336	0.9555	0.6000
(1e7): (1e2) + activation : sigmoid	1,094,314	30	0.6876	0.8982	0.5454
(1e8): (1e2) + activation : tanh	1,094,314	7	0.6949	0.8571	0.5757
(1e9): (1e2) + optimization : SGD	1,094,314	47	0.7723	0.9231	0.7151
(1eA): (1e2) + optimization : RMSprop	1,094,314	7	0.7360	0.8301	0.6484
(1eB): (1e2) + optimization : adadelta	1,094,314	99	0.3026	0.1960	0.1636
(1eC): (1e2) + optimization : Nadam	1,094,314	13	0.7481	0.8894	0.6727
(1eD): (1e2) + data preproccessing	1,094,314	22	0.7917	0.9299	0.6848
(1eE): (1e2) + dimensionality reduction (PCA 0.5)	1,094,314	23	0.8958	0.9412	0.7536
(1eF): (1e2) + dimensionality reduction (PCA 0.7)	1,094,314	14	0.7457	0.9056	0.5696

Στις πρώτες 4 δοκιμές έγιναν αλλαγές στην αρχιτεκτονική του δικτύου, δηλαδή διαφορετικοί αριθμοί επιπέδων και νευρώνων σε κάθε επίπεδο. Το δίκτυο που έδωσε το καλύτερο αποτέλεσμα ήταν αυτό με 4 κρυφά επίπεδα με 256,128,64,32 νευρώνες στο κάθε επίπεδο αντίστοιχα.

Στην συνέχεια πάνω σε αυτή την αρχιτεκτονική επιχειρήθηκαν 4 παραλαγές της συνάρτησης ενεργοποίησης των κρυφών νευρώνων χωρίς να υπάρξει κάποια βελτιστοποίηση στην απόδοση.

Έπειτα δοκιμάστηκαν 3 διαφορετικές συναρτήσεις βελτιστοποίησης με την SGD να δίνει ένα αρκετά ικανοποιητικό αποτέλεσμα αλλά χωρίς να προσφέρει κάποια περαιτέρω βελτιστοποίηση .

Μετά από αυτό επιχειρήθηκε να πραγματοποιηθεί κάποια προ επεξεργασία δεδομένων με την χρήση του ImageDataGenerator και παρόλο που πήραμε και εκεί πολύ καλά αποτελέσματα δεν ξεπεράστηκε ή μέχρι τώρα βέλτιστη απόδοση.

Τέλος χρησιμοποιήθηκε το dimensionality reduction, το οποίο στοχεύει στην μείωση κάποιον διαστάσεων οι οποίες δεν προσφέρουν κάτι στην ταξινόμηση, η μπορεί και να την δυσχεραίνουν σε ορισμένες περιπτώσεις. Ετσί χρησιμοποιώντας τον αλγόριθμο PCA καταφέραμε ένα πάρα πολύ καλό αποτέλεσμα με το validation accuracy να αγγίζει το 0.89. Το βέλτιστο λοιπόν νευρωνικό δίκτυο που προέκυψε από τις συγκεκριμένες δοκιμές είναι το 1eE και έχει την παρακάτω δομή :

```
[95] x_train, x_val, Y_train,Y_val = train_test_split(x,y, test_size=0.2, random_state = 42)
     x_train, x_test, Y_train,Y_test = train_test_split(x_train,Y_train, test_size=0.1, random_state = 42)


     x_train= x_train.reshape(x_train.shape[0],x_train.shape[1]*x_train.shape[2])
     x_val = x_val.reshape(x_val.shape[0],x_val.shape[1]*x_val.shape[2])
     x_test = x_test.reshape(x_test.shape[0],x_test.shape[1]*x_test.shape[2])

     scaler = MinMaxScaler(feature_range=(-1, 1))
     x_train = scaler.fit_transform(x_train)
     x_val = scaler.fit_transform(x_val)
     x_test = scaler.fit_transform(x_test)
```

```
pca = PCA(.5)
pca.fit(x_train)
pca.fit(x_val)
pca.fit(x_test)
```

*ο αλγόριθμος PCA απαιτεί την εγκατάσταση : from sklearn.decomposition import PCA

Layer (type)	Output Shape	Param #
dense_79 (Dense)	(None, 256)	1048832
batch_normalization_58 (Batc	(None, 256)	1024
activation_77 (Activation)	(None, 256)	0
dropout_62 (Dropout)	(None, 256)	0
dense_80 (Dense)	(None, 128)	32896
batch_normalization_59 (Batc	(None, 128)	512
activation_78 (Activation)	(None, 128)	0
dropout_63 (Dropout)	(None, 128)	0
dense_81 (Dense)	(None, 64)	8256
batch_normalization_60 (Batc	(None, 64)	256
activation_79 (Activation)	(None, 64)	0
dropout_64 (Dropout)	(None, 64)	0
dense_82 (Dense)	(None, 32)	2080
batch_normalization_61 (Batc	(None, 32)	128
activation_80 (Activation)	(None, 32)	0
dropout_65 (Dropout)	(None, 32)	0
dense_83 (Dense)	(None, 10)	330
activation_81 (Activation)	(None, 10)	0

Total params: 1,094,314
Trainable params: 1,093,354
Non-trainable params: 960

τα δεδομένα εισόδου εισέρχονται στο πρώτο fully connected κρυφό επίπεδο (256 νευρώνων) , έπειτα περνάνε από batch normalization και στην συνέχεια περνάνε από την συνάρτηση ενεργοποίησης η οποία παρέμεινε η RELU Μετά πραγματοποιούμε ένα dropout στο 20% των νευρώνων και μετά μπαίνει στο επόμενο κρυφό επίπεδο (128) νευρώνων αυτή η διαδικασία επαναλαμβάνεται για όλα τα κρυφά επίπεδα εώς ότου φτάσουμε στο επίπεδο εξόδου όπου θα έχουμε 10 νευρώνες ,δηλαδή όσες είναι οι κλάσεις ταξινόμησης μας . Από εκεί περνάμε τις εξόδους από την softmax συνάρτηση.

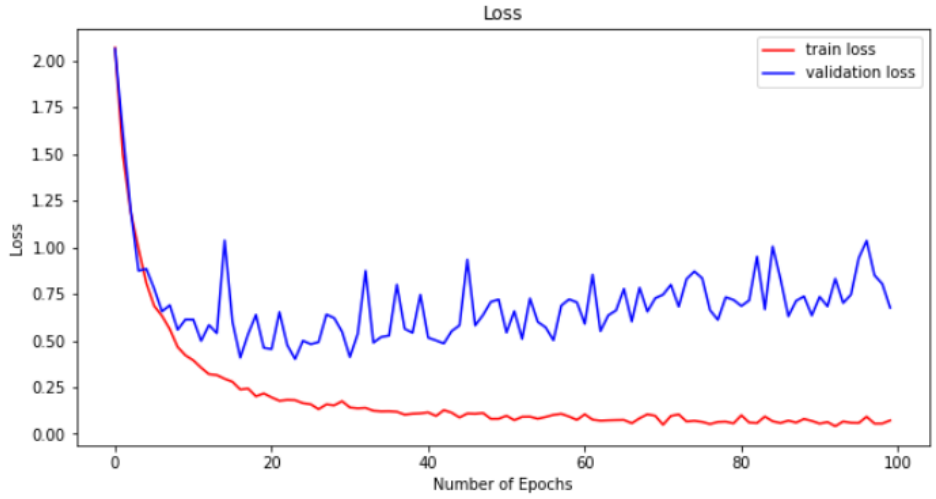
Όσον αφορά την εκπαίδευση θα χρησιμοποιήσουμε έναν optimizer ο οποίος θα έχει τα εξής χαρακτηριστικά:

- Loss function : Categorical crossentropy (χρησιμοποιείται κατά κόρων σε προβλήματα multiclass classification)
- Optimizer: Adam που είναι μία πιο εξελιγμένη μορφή του gradient descent
- Και ορίζουμε στις μετρικές που θέλουμε να πάρουμε το accuracy

Για να εκπαιδεύσουμε το μοντέλο μας χρησιμοποιούμε την εντολή **model.fit** η οποία παίρνει ως παραμέτρους :

- Δεδομένα εκπαίδευση
- Αριθμός εποχών εκπαίδευσης
- Την λίστα που παρακολουθεί την καλύτερη εποχή
- Τα δεδομένα validation
- Και την ρύθμιση για να μπορούμε να έχουμε εικόνα της εξέλιξης του δικτύου (verbose=1)

Παρακάτω φαίνεται και η εξέλιξη της εκπαίδευσης ως προς το Loss



2. Συνελικτικά Νευρωνικά Δίκτυα/Convolutional Neural Networks (CNN)

Ερώτημα 2.a.

Ξεκινήστε χρησιμοποιώντας δυο επίπεδα συνέλιξης με 64 και 32 φίλτρα αντίστοιχα με μέγεθος παραθύρου 3x3, και ένα fully connected επίπεδο με 128 νευρώνες. Μετά από κάθε συνελικτικό επίπεδο χρησιμοποιείστε επίπεδο υποδειγματοληψίας (pooling), προτείνετε max pooling.

Απάντηση 2.a.

Αρχικά το CNN απαιτεί μεγάλο αριθμό δειγμάτων για να είναι αποτελεσματικό, οπότε εμείς εφαρμόζουμε το data augmentation αφού εισάγουμε :

from keras.preprocessing.image import ImageDataGenerator

```
# Changing rotaion, zoom
datagen = ImageDataGenerator(rotation_range=8, width_shift_range=0.08, shear_range=0.3,height_shift_range=0.08, zoom_range=0.08)
new_images = datagen.flow(x_train,batch_size = 1)
datagen.fit(x_train)
```

Έπειτα για διάφορες τιμές pooling παίρνουμε :

Pooling	Παράμετροι	Καλύτερη εποχή	Validation accuracy	Training accuracy
(2,2)	1,069,098	183	0.9757	0.9818
(3,3)	221,226	180	0.9830	0.9770
(5,5)	36,906	199	0.9564	0.9231
(9,9)	86,058	188	0.9757	0.9730

Οι τιμές που προκύπτουν για τα διάφορα μεγέθη pooling είναι πολύ κοντά μεταξύ τους . Ήμουν ανάμεσα στο 3,3 με την καλύτερη απόδοση και τις περισσότερες παραμέτρους και στο 9,9 με τις λιγότερες παραμέτρους και την πολύ κοντά σε αυτό απόδοση αλλά επιλέγω το 3,3.

```
[21] model2b = Sequential()

     model2b.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same', activation ='relu', input_shape = (64,64,1)))
     model2b.add(MaxPool2D(pool_size=(3,3)))
     model2b.add(Dropout(0.2))

     model2b.add(Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same', activation ='relu'))
     model2b.add(MaxPool2D(pool_size=(3,3), strides=(3,3)))
     model2b.add(Dropout(0.2))

     model2b.add(Flatten())
     model2b.add(Dropout(0.2))
     model2b.add(Dense(128, activation = "relu"))
     model2b.add(Dense(10, activation='softmax'))

     model2b.summary()
```

Το δίκτυο θα είναι :

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 64, 64, 64)        640
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 21, 21, 64)        0
_________________________________________________________________
dropout_12 (Dropout)         (None, 21, 21, 64)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 21, 21, 32)        18464
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
dropout_13 (Dropout)         (None, 7, 7, 32)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 1568)              0
_________________________________________________________________
dropout_14 (Dropout)         (None, 1568)              0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               200832
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1290
=================================================================
Total params: 221,226
Trainable params: 221,226
Non-trainable params: 0
_________________________________________________________________
```

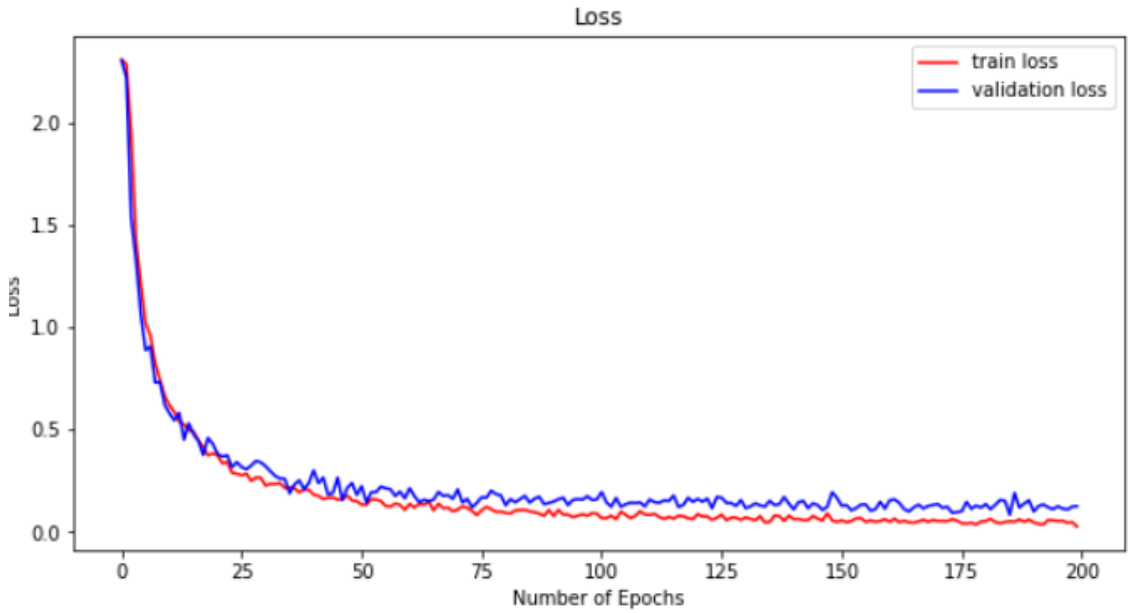
Το δίκτυο μας αποτελείται πέρα από το επίπεδο εισόδου από 2 κρυφά convolutional επίπεδα, με 64 και 32 νευρώνες αντίστοιχα 1 fully connected επίπεδο, με 128 νευρώνες και ένα επίπεδο εξόδου με 10 νευρώνες

Τα δεδομένα εισόδου αφού περάσουν από το πρώτο συνελικτικό επίπεδο πέρανε από ένα επίπεδο pooling το οποίο έχει σκοπό να μειώσει σταδιακά το χωρικό μέγεθος της αναπαράστασης για να μειώσει την ποσότητα παραμέτρων και υπολογισμού στο δίκτυο. Έπειτα γίνεται το dropout όπως εξηγήθηκε παραπάνω. Αυτό επαναλαμβάνεται και για το δεύτερο συνεκτικό επίπεδο. Η έξοδος του δεύτερου συνελικτικού περνάει από την συνάρτηση flatten για να μπορέσει να την υποδεχτεί το dence επίπεδο.

Ερώτημα 2.b.

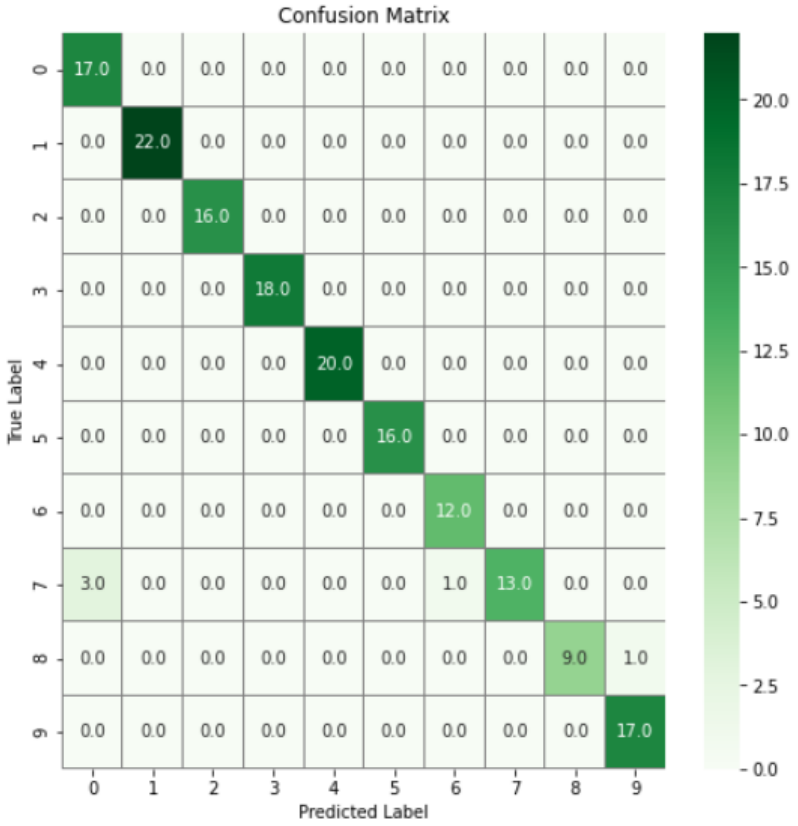
Παραγωγή των γραφημάτων train/test – loss ανά εποχή. Εμφανίζεται το πρόβλημα underfitting/overfitting; Αν ναι, να προτείνετε έναν τρόπο διαχείρισης του φαινομένου. Επίσης να αιτιολογήσετε ποιος είναι ο βέλτιστος αριθμός εποχών εκπαίδευσης. Για τον βέλτιστο αριθμό εποχών εκπαίδευσης, ζητείται να εξαχθούν οι μετρικές: accuracy, precision recall, f1 score, confusion matrix.

Απάντηση 2.b.




```
best epoch:  185
best epoch validation accuracy:  0.9854721426963806
best epoch train accuracy:  0.9838274717330933
best epoch precision:  0.9846002459526062
best epoch recall:  0.9820478558540344
best epoch f1:  0.983292818069458
```

Για τις εποχές που εξετάζουμε δεν παρατηρείται το πρόβλημα του overfitting προς το παρόν. Καθώς το validation accuracy δεν έχει χειροτερέψει σημαντικά για να πούμε ότι σίγουρα δεν θα δώσει κάτι καλύτερο αν το αφήσουμε για παραπάνω εποχές



Βλέπουμε ότι συγκριτικά με πριν τα αποτελέσματα είναι πολύ καλύτερα χωρίς να έχουμε κάνει κάποια παραπάνω δουλεία εμείς πάνω στο dataset (αντιθέτως το διαστρεβλώσαμε παραπάνω)

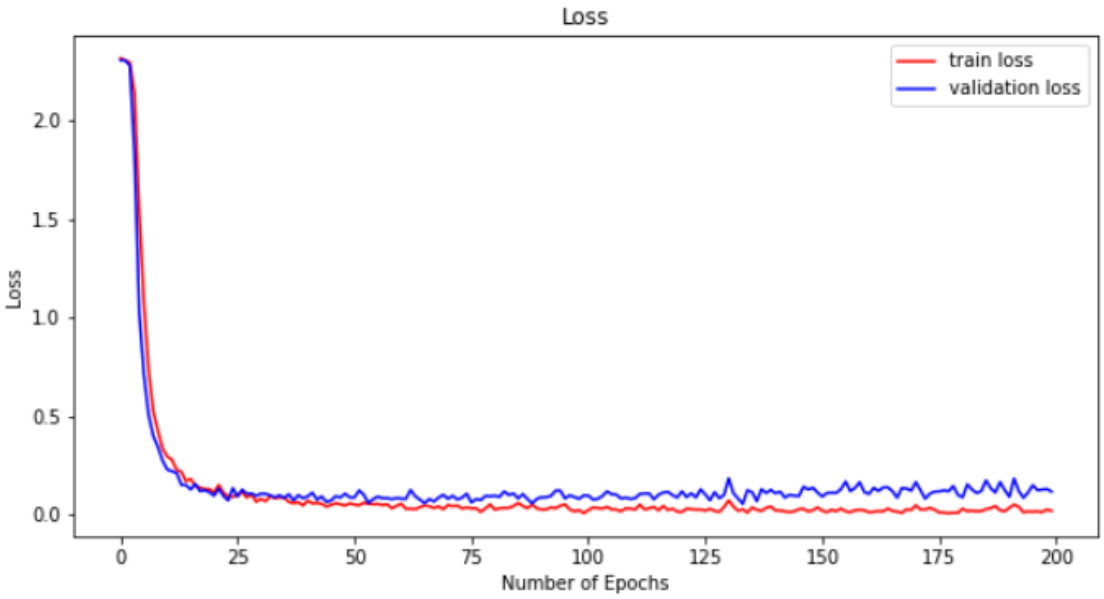
Ερώτημα 2.c.

Υλοποιείστε διάφορες δομές συνελικτικών τεχνητών νευρωνικών δικτύων και μεταβλητού αριθμού νευρώνων.

Απάντηση 2.c.

Παραλλαγή	Παράμετροι	Εποχή	Validation accuracy	Training accuracy	Test accuracy
(2c1): 2 convolutional (32,64), 1 fully connected(128)	872,202	180	0.9903	0.9946	0.9818
(2c2): 2 convolutional (32,64), 2 fully connected(128,64)	879,818	149	0.9903	0.9912	0.9757
(2c3): 4 convolutional (32,32,64,64), 2 fully connected(64,128)	177,066	178	0.9878	0.9858	0.9757
(2c4): 3 convolutional (32,64,128), 1 fully connected(256)	226,570	53	0.9830	0.97439	0.9818
(2c5): 3 convolutional (32,64,128), 1 fully connected(128)	372,618	133	0.9975	0.9925	0.9757
(2c6): 4 convolutional (32,64,128,256), 1 fully connected(256)	1,077,386	192	0.9951	0.9838	0.9696
(2c7): 3 convolutional (32,64,128), 1 fully connected(64)	298,186	144	0.9927	0.9885	0.9818
(2c8): 2 convolutional (16,32,64), 1 fully connected(128)	57,482	162	0.9878	0.9743	0.9696
(2c9): 3 convolutional (16,32,64), 1 fully connected(256)	91,658	119	0.9903	0.9710	0.9696
(2cA): 4 convolutional (16,32,32,64), 1 fully connected(128)	42,154	160	0.9903	0.9649	0.9818

Γενικά είναι πολύ κοντά στην απόδοση τους τα δίκτυα . Επιλέγω σαν καλύτερο αυτό με το καλύτερο validation accuracy αλλά θα μπορούσε να είναι κα το 2cA το οποίο με πολύ λιγότερες παραμέτρους πετυχαίνει ένα πολύ κοντινό αποτέλεσμα



Βλέπουμε και εδώ ότι δύσκολα αποφασίζουμε αν υπάρχει overfit αφού το validation error δεν έχει χειροτερεύσει αισθητά και μπορεί για περισσότερες εποχές να έδινε κάτι καλύτερο.

Ερώτημα 2.d.

CNN tuning: Χρησιμοποιείστε κανονικοποίηση των δεδομένων σας (batch normalization). Τι επίδραση έχει στην ακρίβεια;

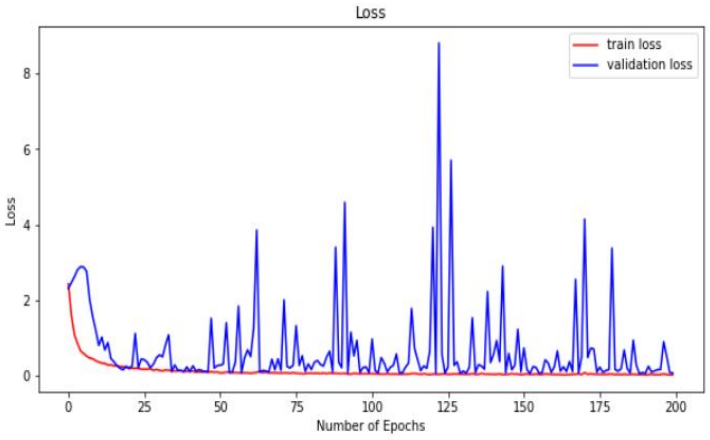
Απάντηση 2.d.

```
model2d = Sequential()

model2d.add(Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same', input_shape = (64,64,1)))
model2d.add(BatchNormalization())
model2d.add(Activation('relu'))
model2d.add(MaxPool2D(pool_size=(3,3)))
model2d.add(Dropout(0.2))

model2d.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same'))
model2d.add(BatchNormalization())
model2d.add(Activation('relu'))
model2d.add(MaxPool2D(pool_size=(3,3), strides=(3,3)))
model2d.add(Dropout(0.2))

model2d.add(Conv2D(filters = 128, kernel_size = (3,3),padding = 'Same'))
model2d.add(BatchNormalization())
model2d.add(Activation('relu'))
model2d.add(MaxPool2D(pool_size=(3,3), strides=(3,3)))
model2d.add(Dropout(0.2))

model2d.add(Flatten())
model2d.add(Dropout(0.2))
model2d.add(Dense(128))
model2d.add(Activation('relu'))
model2d.add(BatchNormalization())

model2d.add(Dense(10, activation='softmax'))

model2d.summary()
```

Layer (type)	Output Shape	Param #
conv2d_41 (Conv2D)	(None, 64, 64, 32)	320
batch_normalization (BatchNo	(None, 64, 64, 32)	128
activation (Activation)	(None, 64, 64, 32)	0
max_pooling2d_41 (MaxPooling	(None, 21, 21, 32)	0
dropout_56 (Dropout)	(None, 21, 21, 32)	0
conv2d_42 (Conv2D)	(None, 21, 21, 64)	18496
batch_normalization_1 (Batch	(None, 21, 21, 64)	256
activation_1 (Activation)	(None, 21, 21, 64)	0
max_pooling2d_42 (MaxPooling	(None, 7, 7, 64)	0
dropout_57 (Dropout)	(None, 7, 7, 64)	0
conv2d_43 (Conv2D)	(None, 7, 7, 128)	73856
batch_normalization_2 (Batch	(None, 7, 7, 128)	512
activation_2 (Activation)	(None, 7, 7, 128)	0
max_pooling2d_43 (MaxPooling	(None, 2, 2, 128)	0
dropout_58 (Dropout)	(None, 2, 2, 128)	0
flatten_15 (Flatten)	(None, 512)	0
dropout_59 (Dropout)	(None, 512)	0
dense_32 (Dense)	(None, 128)	65664
activation_3 (Activation)	(None, 128)	0
batch_normalization_3 (Batch	(None, 128)	512
dense_33 (Dense)	(None, 10)	1290

Total params: 161,034
Trainable params: 160,330
Non-trainable params: 704

```
best epoch:  188
best epoch validation accuracy:  0.9878934621810913
best epoch train accuracy:  0.9939352869987488
```




```
6/6 [==============================] - 0s 7ms/step - loss: 0.2848 - accuracy: 0.9818
Test score:  0.2848461866378784
Test accuracy:  0.9818181991577148
```

Ερώτημα 2.e.

Δοκιμάστε αρχιτεκτονική CNN χωρίς fully connected layer. Μπορεί να δουλέψει; Ακολούθως χρησιμοποιείστε συνελικτικά επίπεδα χωρίς ενδιάμεσα επίπεδα υποδειγματοληψίας (pooling).

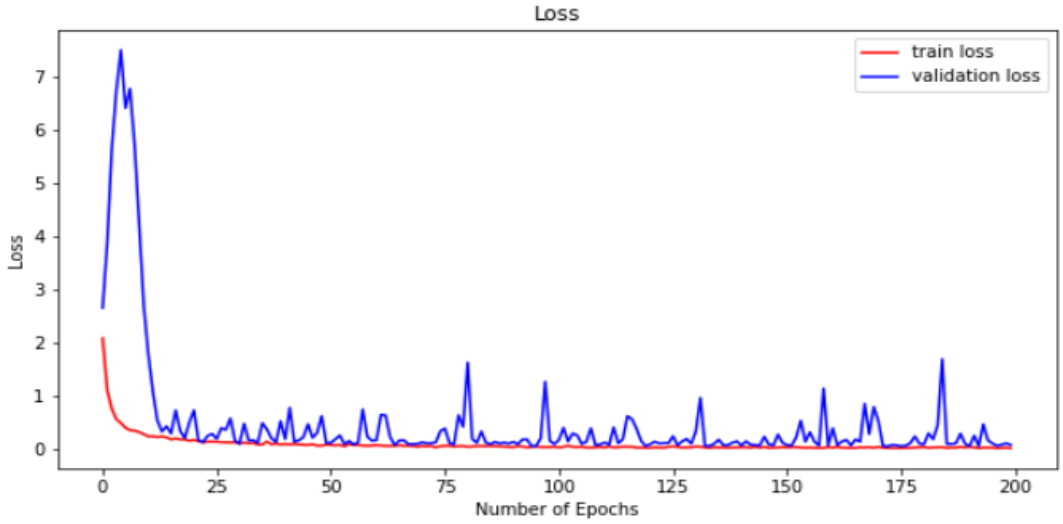
Απάντηση 2.e.

Layer (type)	Output Shape	Param #
conv2d_44 (Conv2D)	(None, 64, 64, 32)	320
batch_normalization_4 (Batch	(None, 64, 64, 32)	128
activation_4 (Activation)	(None, 64, 64, 32)	0
max_pooling2d_44 (MaxPooling	(None, 21, 21, 32)	0
dropout_60 (Dropout)	(None, 21, 21, 32)	0
conv2d_45 (Conv2D)	(None, 21, 21, 64)	18496
batch_normalization_5 (Batch	(None, 21, 21, 64)	256
activation_5 (Activation)	(None, 21, 21, 64)	0
max_pooling2d_45 (MaxPooling	(None, 7, 7, 64)	0
dropout_61 (Dropout)	(None, 7, 7, 64)	0
conv2d_46 (Conv2D)	(None, 7, 7, 128)	73856
batch_normalization_6 (Batch	(None, 7, 7, 128)	512
activation_6 (Activation)	(None, 7, 7, 128)	0
max_pooling2d_46 (MaxPooling	(None, 2, 2, 128)	0
dropout_62 (Dropout)	(None, 2, 2, 128)	0
conv2d_47 (Conv2D)	(None, 2, 2, 256)	295168
activation_7 (Activation)	(None, 2, 2, 256)	0
max_pooling2d_47 (MaxPooling	(None, 1, 1, 256)	0
batch_normalization_7 (Batch	(None, 1, 1, 256)	1024
conv2d_48 (Conv2D)	(None, 1, 1, 10)	2570
activation_8 (Activation)	(None, 1, 1, 10)	0
flatten_16 (Flatten)	(None, 10)	0

Total params: 392,330
Trainable params: 391,370
Non-trainable params: 960

Όπως αναφέρεται και στην βιβλιογραφία, κάθε fully connected layer μπορεί να αντικατασταθεί από ένα convolutional layer. Αυτή είναι και η μέθοδος που χρησιμοποιήθηκε για την πραγματοποίηση του συγκεκριμένου ερωτήματος. Έχουν αντικατασταθεί τα 2 τελευταία Dense layers του ANN από 2 συνελικτικά layers, που έχουν ισοδύναμη λειτουργία.

Στο τέλος προκειμένου να μετατρέψουμε το layer με shape = (1,1,10) σε layer με shape = (10) χρησιμοποιήθηκε ένα Flatten layer. Βλέπουμε ότι μπορεί να λειτουργήσει και πολύ δίνει και καλά αποτελέσματα .




```
best epoch:  172
best epoch validation accuracy:  0.9927361011505127
best epoch train accuracy:  0.9959568977355957
6/6 [==============================] - 0s 8ms/step - loss: 0.3123 - accuracy: 0.9636
Test score:  0.31226563453674316
Test accuracy:  0.9636363387107849
```

```
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_49 (Conv2D)           (None, 64, 64, 32)        320
_________________________________________________________________
batch_normalization_8 (Batch (None, 64, 64, 32)        128
_________________________________________________________________
activation_9 (Activation)    (None, 64, 64, 32)        0
_________________________________________________________________
dropout_63 (Dropout)         (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 64, 64, 64)        131136
_________________________________________________________________
batch_normalization_9 (Batch (None, 64, 64, 64)        256
_________________________________________________________________
activation_10 (Activation)   (None, 64, 64, 64)        0
_________________________________________________________________
dropout_64 (Dropout)         (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_51 (Conv2D)           (None, 64, 64, 128)       8320
_________________________________________________________________
batch_normalization_10 (Batc (None, 64, 64, 128)       512
_________________________________________________________________
activation_11 (Activation)   (None, 64, 64, 128)       0
_________________________________________________________________
dropout_65 (Dropout)         (None, 64, 64, 128)       0
_________________________________________________________________
conv2d_52 (Conv2D)           (None, 64, 64, 256)       33024
_________________________________________________________________
activation_12 (Activation)   (None, 64, 64, 256)       0
_________________________________________________________________
batch_normalization_11 (Batc (None, 64, 64, 256)       1024
_________________________________________________________________
conv2d_53 (Conv2D)           (None, 64, 64, 10)        2570
_________________________________________________________________
activation_13 (Activation)   (None, 64, 64, 10)        0
_________________________________________________________________
flatten_17 (Flatten)         (None, 40960)             0
=================================================================
Total params: 177,290
Trainable params: 176,330
Non-trainable params: 960
_________________________________________________________________
```

Χωρίς pooling όμως δεν γίνεται να έχουμε αποτέλεσμα γιατί δεν γίνεται να συρρικνώσουμε το νευρωνικό δίκτυο οποτε δεν μπορούμε να έχουμε στην έξοδο 10 νευρώνες.

3. Ερωτήσεις Κατανόησης

Ερώτημα 3.a.

Για ποιον λόγο τα CNN έχουν μεγαλύτερη ακρίβεια;

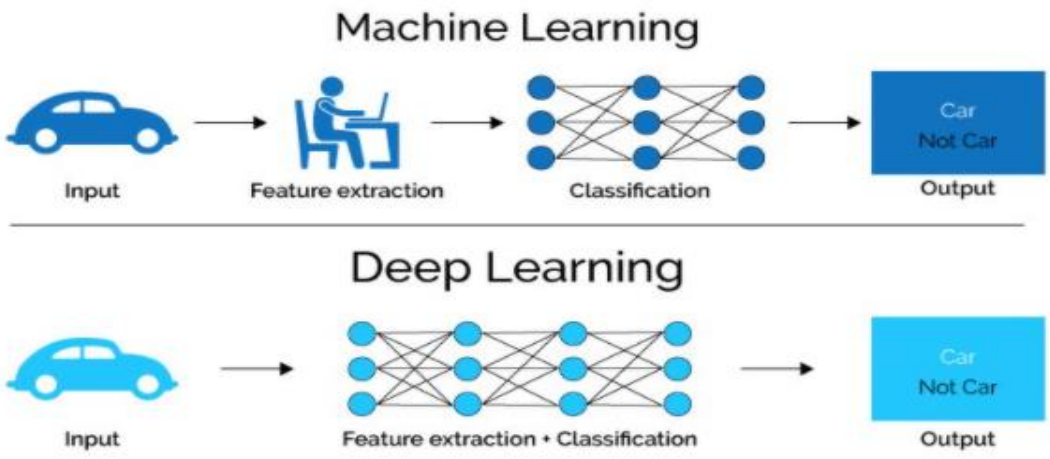
Απάντηση 3.a.

Τα cnn πετυχαίνουν μεγαλύτερη ακρίβεια, λόγω του ότι εκμεταλλεύονται τα χωρικά χαρακτηριστικά που παίρνουν μέσω της συνέλιξης με πολλά feature maps. Έτσι αυτό είναι ιδανικό όταν πρέπει να εξαχθούν χιλιάδες χαρακτηριστικά (όπως στην περίπτωση των εικόνων) Βέβαια, αυτό έχει αντίκτυπο στις συνολικές παραμέτρους του δικτύου, καθώς αυτές αυξάνονται εμφανώς.

Ερώτημα 3.b.

Ποια είναι η ειδοποιός διαφορά των CNN και των ANN ως προς την ταξινόμηση εικόνων;

Απάντηση 3.b.



Στα CNN η διαδικασία εξαγωγής χαρακτηριστικών τα οποία θα μπορούν να χρησιμοποιηθούν μετέπειτα για την καλύτερη ταξινόμησή είναι δουλειά του ίδιου του νευρωνικού δικτύου και όχι του ερευνητή όπως θα ήταν στα artificial neural networks. Με άλλα λόγια η διαδικασία εξαγωγής χαρακτηριστικών για την ταξινόμησή η οποία, ειδάλλως, θα απαιτούσε πολύ καλή γνώση του αντικειμένου που μελετάται και πολύ χρόνο για να πραγματοποιηθεί , τώρα αυτοματοποιείται με την χρήση των CNN

4. Bonus

Ερώτημα 4.a.

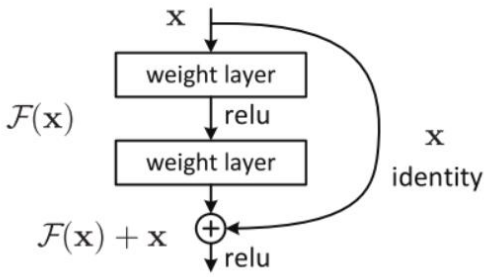
Έχουν παρατεθεί και 3 επιπλέον dataset. Επιλέξτε ένα από αυτά και εκτελέστε το πείραμα με την βέλτιστη αρχιτεκτονική CNN που προτείνατε για το αρχικό dataset. Ταυτόχρονα υλοποιείστε την αρχιτεκτονική ResNet-50.

Ερώτημα 4.b.

Ποιες είναι οι διαφορές ενός CNN με το ResNet;

Απάντηση 4.b.

Τα Resnet δίκτυα επιλύουν το πρόβλημα vanishing gradient με συντομεύσεις ή παραλείψεις συνδέσεων, συνδέοντας ρηχά στρώματα σε βαθιά στρώματα. Σε ένα τέτοιο δίκτυο μπορούμε να στοιβάζουμε μπλοκ όλο και περισσότερο, χωρίς υποβάθμιση της απόδοσης. Αυτό επιτρέπει την κατασκευή πολύ βαθιών δικτύων.Η βασική ιδέα του ResNet είναι η εισαγωγή μιας λεγόμενης «“identity shortcut connection”» που παραλείπει ένα ή περισσότερα επίπεδα, όπως φαίνεται στο παρακάτω σχήμα:



Τα τυπικά μοντέλα ResNet εφαρμόζονται με παραλείψεις διπλού ή τριπλού επιπέδου που περιέχουν μη γραμμικότητες (ReLU) και ομαλοποίηση παρτίδων στο μεταξύ. [1] [2] Μια επιπλέον μήτρα βάρους μπορεί να χρησιμοποιηθεί για την εκμάθηση των βαρών παράλειψης

Αυτό καταλαβαινουμε ότι επιτρέπει στα Resnet (σε αντίθεση με τ CNNs να εμβαθύνουν πολύ και να δίνουν πολύ καλά αποτελέσματα). Αυτό σημαίνει ότι τα RESnet μπορούν να έχουν εκατομμύρια παραμέτρους >20 εκατομμύρια

Ερώτημα 4.c.

Ποιο ήταν το κύριο πρόβλημα που αποφεύχθηκε κατά την εκπαίδευση βαθέων νευρωνικών δικτύων με το ResNet;

Απάντηση 4.c.

Υπήρχε το πρόβλημα του vanishing gradient στα πολύ βαθιά δίκτυα δηλαδή δεν μπορεί να υπάρχει κάποιο update στο δίκτυο και δεν μπορεί να εκπαιδευτεί.

Το κίνητρο για παράβλεψη στρωμάτων είναι να αποφευχθεί το πρόβλημα της vanishing gradient, επαναχρησιμοποιώντας τις ενεργοποιήσεις από ένα προηγούμενο στρώμα έως ότου το παρακείμενο στρώμα μάθει τα βάρη του. Κατά τη διάρκεια της προπόνησης, τα βάρη προσαρμόζονται για ενίσχυση του στρώματος που είχε παραλειφθεί προηγουμένως.

Καθώς προχωράει το νευρωνικό, με άλλα λόγια, θα μπορούσα να προσθέτω και την είσοδο που έχω στην έξοδο του κάθε επιπέδου ώστε να ενισχύεται η έξοδος , οπότε αν δεν προσφέρει κάτι στην ουσία με αυτή την παρεμβολή «υπενθυμίζει» την είσοδο που είχε ώστε να συνεχίζει να την βελτιώνει.