Software Development for Algorithmic Projects

Project 2

Στοιχεία Φοιτητών

Λάζαρος Αυγερίδης - 1115201600013
Θέμης Βαρβέρης - 1115201600015

Github: https://github.com/themisvr/Image-Neural-Network-Autoencoding

Σύντομη Περιγραφή Προγράμματος

Autoencoder

Το πρώτο κομμάτι της εργασίας αφορά την κατασκευή ενός νευρωνικού δικτύου αυτοκωδικοποίησης εικόνων για το MNIST dataset. Ο χρήστης έχει την δυνατότητα να πραγματοποιήσει πειράματα εκπαίδευσης του δικτύου με διαφορετικές τιμές υπερπαραμέτρων (αριθμού συνελικτικών στρωμάτων, μεγέθους συνελικτικών φίλτρων, αριθμού συνελικτικών φίλτρων ανά στρώμα, αριθμού εποχών εκπαίδευσης (epochs), μεγέθους δέσμης (batch size). Το πρόγραμμα είναι πλήρως παραμετροποιημένο ως προς τις παραπάνω υπερπαραμέτρους, καθώς δίνει στον χρήστη την ευχέρεια να τις δώσει για κάθε συνελικτικό επίπεδο (convolutional layer). Επίσης χρησιμοποιούνται και MaxPooling layers (συγκεκριμένα 2 στο πλήθος, λόγω των περιορισμένων διαστάσεων που έχουν οι εικόνες του dataset, 28x28). Σκοπός είναι η ελαχιστοποίηση του σφάλματος (loss), αποφεύγοντας την υπερπροσαρμογή (overfitting). Το πρόγραμμα σχεδιάζει γραφικές παραστάσεις του σφάλματος στο σύνολο εκπαίδευσης και στο σύνολο επικύρωσης ως προς τις τιμές των υπερπαραμέτρων.

Classifier

Στο δεύτερο κομμάτι της εργασίας, κάνουμε “extract” το encoder τμήμα της αρχιτεκτονικής του autoencoder που κατασκευάστηκε και αποθηκεύτηκε στο (Α), του προσθέτουμε κάποια επιπλέον layers (πχ Fully-connected, Output με softmax) και έτσι κατασκευάζουμε ένα νέο νευρωνικό δίκτυο το οποίο είναι πλέον ένας κατηγοριοποιητής εικόνων.
Το πρόγραμμα δίνει τη δυνατότητα στο χρήστη να πραγματοποιήσει πολλαπλά πειράματα εκπαιδεύοντας περισσότερα του ενός CNN’s models. Για καθένα από αυτά τα πειράματα, το πρόγραμμα δίνει εικόνα στο χρήστη σχετικά με το πόσο καλά γενικεύει καθένα από αυτά τα μοντέλα (accuracy, loss). Στη συνέχεια, και μετά από έναν αριθμό τέτοιων πειραμάτων το πρόγραμμα δίνει τη δυνατότητα στο χρήστη είτε να σχεδιάσει γραφικές παραστάσεις συγκεντρωτικά για όλα τα πειράματα που εκτελέστηκαν, ή να εμφανίσει ένα δείγμα των εικόνων από το σύνολο ελέγχου, όπου για την κάθε μία από αυτές θα μπορεί να δει την κατηγορία στην οποία το νευρωνικό δίκτυο προέβλεψε ότι ανήκουν και στην κατηγορία στην οποία ανήκουν πραγματικά.
Σε κάθε πείραμα, το πρόγραμμα ζητά από το χρήστη τιμές για τις παρακάτω υπερ παραμέτρους:

1. αριθμό κόμβων στο fully-connected layer
2. αριθμό epochs (για το 1ο στάδιο εκπαίδευσης + για το 2ο στάδιο εκπαίδευσης)
3. batch size (για το 1ο στάδιο εκπαίδευσης + για το 2ο στάδιο εκπαίδευσης)

Κατάλογος Αρχείων Κώδικα

Το repository περιέχει 4 καταλόγους.

- **src:** Ο κατάλογος αυτός περιέχει όλα τα αρχεία κώδικα (.py αρχεία). Πιο συγκεκριμένα, βρίσκονται τα εξής αρχεία.
    - **utils.py:** Περιέχει συναρτήσεις χρήσιμες για I/O λειτουργίες, όπως διάβασμα των datasets, συναρτήσεις για το parsing των command line arguments και συναρτήσεις για το user interface με το οποίο θα αλληλεπιδρά ο χρήστης.
    - **cnn_model.py:** Περιέχει την κλάση **Autoencoder,** η οποία προσωμοιώνει το μοντέλο του autoencoder και υλοποιεί μεθόδους απαραίτητες για την δημιουργία του convolutional neural network.
    - **autoencoder.py:** Αυτό είναι το αρχείο κώδικα που θα εκτελέσει ο χρήστης (ερώτημα Α) για την εκπαίδευση του μοντέλου.

    - **cnn_classifier.py:** Περιέχει την κλάση **Classifier,** η οποία προσωμοιώνει το μοντέλο του classifier και υλοποιεί μεθόδους απαραίτητες για την εκπαίδευση του convolutional neural network αλλά και την αποτίμηση των αποτελεσμάτων του.
    - **classifier.py:** Αυτό είναι το αρχείο που θα εκτελέσει ο χρήστης (ερώτημα Β).
- **datasets:** Ο κατάλογος αυτός περιέχει αρχεία σχετικά με τα train-images και test-images, αλλά και τα αρχεία train-labels και test-labels (όπου θα χρησιμοποιηθούν στο Β μέρος του project, τον classifier).
- **saved_cnns:** Ο κατάλογος αυτός περιέχει τα pre-trained μοντέλα που είτε αποθηκεύσαμε εμείς, είτε αποθηκεύει ο χρήστης κατά την εκτέλεση των προγραμμάτων. Τα αρχεία αυτά έχουν την εξής μορφή: “<model-name>.h5”.
- **results:** Ο κατάλογος αυτός περιέχει αρχεία που απεικονίζουν γραφήματα σχετικά με τα αποτελέσματα του **autoencoder** και του **classifier.**

Οδηγίες Χρήσης

Autoencoder

Η εκτέλεση για τον autoencoder γίνεται ως εξής:
(python version 3.6 or above)

```
$ python3 autoencoder.py -d <dataset>
```

Όταν το πρόγραμμα αρχίσει να εκτελείται συμβαίνουν οι εξής λειτουργίες:

1. Διαβάζουμε το αρχείο <dataset> που δίνει ο χρήστης στην γραμμή εντολής.
2. Ζητάμε από τον χρήστη εάν θέλει να φορτώσουμε κάποιο υπάρχων pretrained μοντέλο.
    a. Στην περίπτωση που το ζητήσει, τότε του ζητάμε να μας δώσει το path στο οποίο βρίσκεται το μοντέλο αυτό. Ζητάμε επίσης από τον χρήστης να μας δώσει τον αριθμό των epochs και του batch size με τα οποία θα γίνει train το μοντέλο μας. Έπειτα, χωρίζουμε το dataset σε trainSet και validationSet και κάνουμε κανονικοποίηση των διαστάσεων των εικόνων (συγκεκριμένα του MNIST dataset). Στην συνέχεια φορτώνουμε το μοντέλο από το δοσμένο μονοπάτι, κάνουμε train και εκτυπώνουμε γραφήματα με τα αποτελέσματα της εκπαίδευσης.

3. Στην συνέχεια εκτελείται το παρακάτω κομμάτι κώδικα για την εκπαίδευση του autoencoder.

```
def cnn_train_simulation():
    epochs, batch_size, convs = ask_for_hyperparameters()

    autoencoder = Autoencoder(dataset, dims, epochs, batch_size, convs)
    autoencoder._split_dataset(testSize)
    autoencoder._reshape()

    input_img = keras.Input(shape=(rows, cols, 1))
    encoded = autoencoder.encoder(input_img)
    decoded = autoencoder.decoder(encoded)
    autoencoder.compile_model(input_img, decoded)
    autoencoder.train_model()

    return autoencoder
```

4. Τέλος εκτελείται η συνάρτηση menu(), η οποία αλληλεπιδρά με τον χρήστη δίνοντάς του της εξής επιλογές:

a. Εάν θέλει να επαναλάβει το πείραμα με διαφορετικές υπερπαραμέτρους. Στην περίπτωση αυτή ακολουθούνται τα βήματα του κώδικα παραπάνω για την εκπαίδευση του νέου μοντέλου.
b. Εάν θέλει να εκτυπωθούν τα γραφήματα για το loss, val_loss ως προς τις υπερπαραμέτρους που έδωσε για το πείραμα αυτό.
c. Εάν θέλει να αποθηκεύσει το μοντέλο που μόλις εκπαιδεύτηκε. Στην περίπτωση αυτή του ζητείται να δώσει το μονοπάτι που θα το αποθηκεύσει.
d. Εάν θέλει να πραγματοποιήσει έξοδο του προγράμματος.

Classifier

Η εκτέλεση για το *classification.py* γίνεται ως εξής:
(python version 3.6 or above)

```
$ python3 classification.py -d <training set> -dl <training labels>
                            -t <test set> -tl <test labels> -model <autoencoder.h5>
```

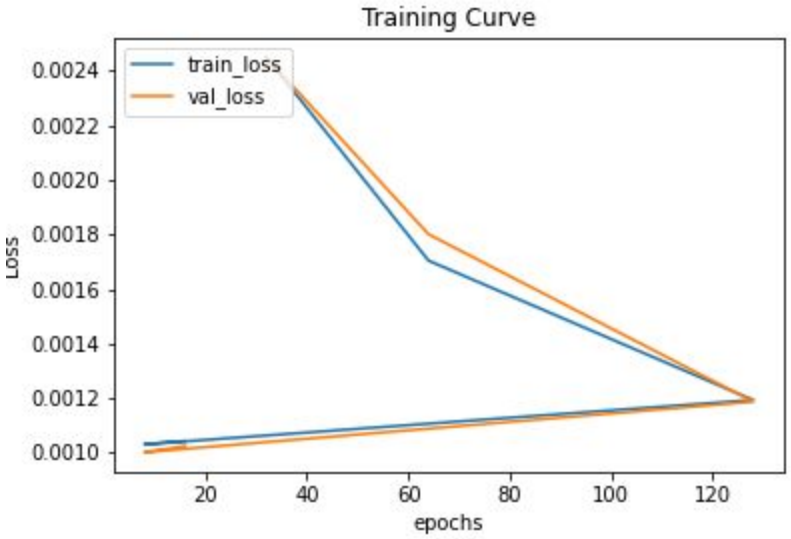
Προεπεξεργασία: Αρχικά το πρόγραμμα διαβάζει από τα αρχεία <training set> και <training labels> τα δεδομένα εκπαίδευσης και τις κλάσεις / ετικέτες τους αντίστοιχα. Όμοια, διαβάζει από τα αρχεία <test set> και <test labels> τα δεδομένα ελέγχου και τις κλάσεις / ετικέτες τους αντίστοιχα. Έπειτα, για τα training και test set δεδομένα (εικόνες) κάνουμε κανονικοποίηση των pixels τους στο [0, 1] και μετά τα μετατρέπουμε σε 4D tensors με shape (total samples, rows, cols, input channels), δηλαδή σε μορφή αποδεκτή από τις συναρτήσεις του keras *fit()*, *predict()*, κλπ. Τέλος, “σπάμε” το training set σε train και validation sets χρησιμοποιώντας ποσοστό 10%.
Φόρτωση Encoder Αρχιτεκτονικής: Μετά το preprocessing των δεδομένων, φορτώνεται το autoencoder model από το path που έδωσε ο χρήστης στη γραμμή εντολών. Από την αρχιτεκτονική του autoencoder, θα κρατήσουμε μόνο το τμήμα του encoder.
Κατασκευή CNN, εκπαίδευση και αξιολόγηση: Ακολουθεί το training loop του προγράμματος κατά το οποίο ο χρήστης εκτελεί πειράματα. Πιο συγκεκριμένα για κάθε πείραμα:

1. **Εισάγει μέσω stdin τον αριθμό των κόμβων που θα έχει το fully-connected layer του δικτύου**. Μετά από αυτήν την επιλογή του χρήστη, κατασκευάζουμε το νέο νευρωνικό δίκτυο (CNN), το οποίο χρησιμοποιεί την αρχιτεκτονική του encoder (προσθέτοντας μετά από κάθε max pooling layer ένα dropout layer) και στο τέλος αυτής έχουν προστεθεί με την εξής σειρά: Flatten, Dense (με τον αριθμό κόμβων που έδωσε ο χρήστης), Batch Normalization, Dropout, Dense (το output layer με softmax activation function και 10 outputs).
2. Αφού κατασκευαστεί το CNN του πειράματος, προχωράμε στην εκπαίδευση του δικτύου η οποία πραγματοποιείται σε 2 στάδια. Στο 1^{ο} στάδιο γίνεται εκπαίδευση μόνο των παραμέτρων (weights) του fully-connected layer, ενώ στο 2^{ο} στάδιο γίνεται εκπαίδευση των παραμέτρων ολόκληρου του δικτύου, το οποίο προφανώς είναι το τμήμα του προγράμματος που απαιτεί τον περισσότερο χρόνο. **Σε κάθε στάδιο εκπαίδευσης, ο χρήστης εισάγει μέσω stdin τον αριθμό των epochs και το batch size**. (για τα plots χρησιμοποιείται η τιμή των epochs και του batch size του 2^{ου} σταδίου εκπαίδευσης)

3. Μετά την εκπαίδευση, γίνεται η αξιολόγηση του δικτύου χρησιμοποιώντας το σύνολο ελέγχου (test set). Στο σημείο αυτό εκτυπώνονται στο χρήστη το test accuracy και το test loss.
4. Ο χρήστης τώρα έχει 3 επιλογές:
    a. Να προχωρήσει σε διεξαγωγή νέου πειράματος, δηλαδή σε κατασκευή, εκπαίδευση και αξιολόγηση ενός νέου δικτύου. Στην περίπτωση αυτή η ροή εκτέλεσης του προγράμματος επιστρέφει στο βήμα (1).
    b. Να εμφανίσει γραφικές παραστάσεις για το training + validation loss και για το training + validation accuracy για **το σύνολο των πειραμάτων που εκτελέστηκαν**. Επιπλέον για κάθε πείραμα εκτυπώνει στο χρήστη ένα classification report. Να σημειωθεί εδώ, ότι για να εκτελεστεί αυτή η ενέργεια, **θα πρέπει στα προηγούμενα πειράματα ο χρήστης να έχει κρατήσει 2 από τις υπερ παραμέτρους σταθερές, και να έχει μεταβάλλει μόνο την 3η**. Σε αντίθετη περίπτωση, το πρόγραμμα τερματίζει με μήνυμα λάθους.
    c. Να κάνει οπτικοποίηση ενός δείγματος των εικόνων του test set και την κατηγορία στην οποία πρόβλεψε το νευρωνικό δίκτυο ότι ανήκουν. Αν ο χρήστης έχει κάνει περισσότερα του ενός πειράματα, τότε του δίνεται η δυνατότητα να επιλέξει το δίκτυο, του οποίου οι προβλέψεις θα οπτικοποιηθούν. Ακόμα, εκτυπώνεται ο αριθμός των εικόνων που κατηγοριοποιήθηκαν σωστά και ο αριθμός αυτών που κατηγοριοποιήθηκαν λανθασμένα.

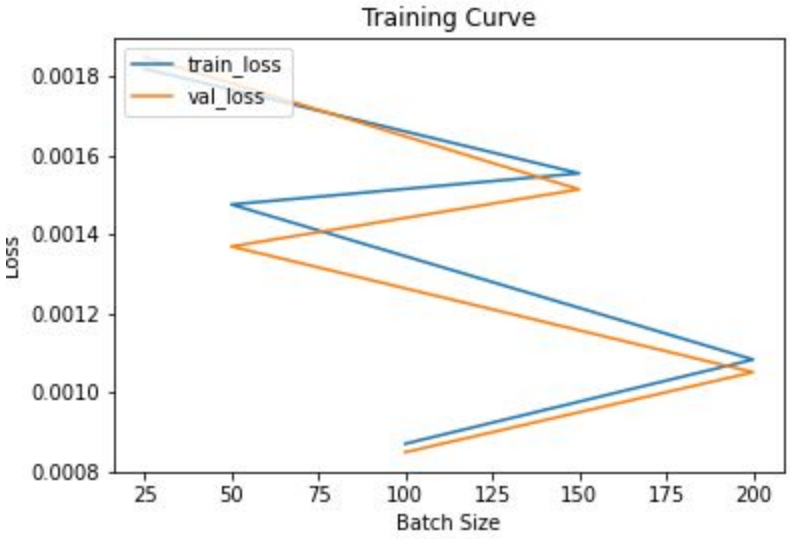
Αποτελέσματα

Autoencoder

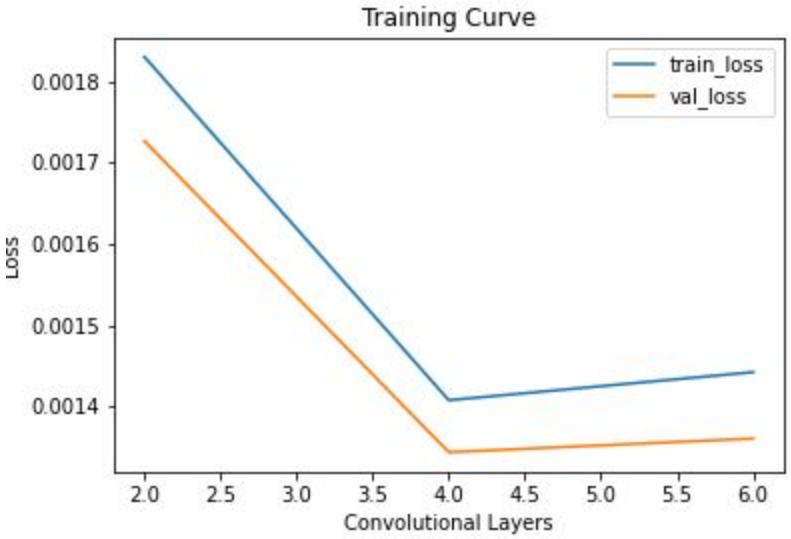


1. Convolutional Layers = 2

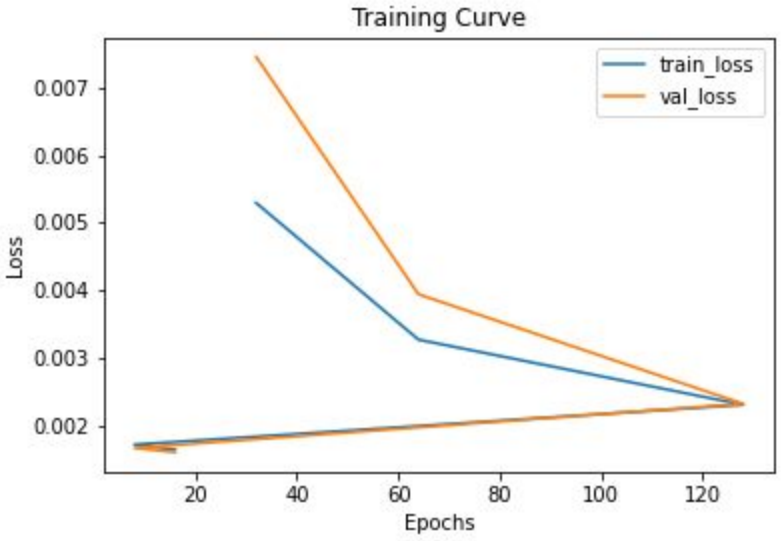
    a. Conv1: #filters = 32, kernel_size = 3
    b. MaxPooling Layer (2, 2)
    c. Conv2: #filters = 64, kernel_size = 3
    d. MaxPooling Layer (2, 2)
2. Batch Size = 50
3. Epochs = [8, 16, 32, 64, 128]



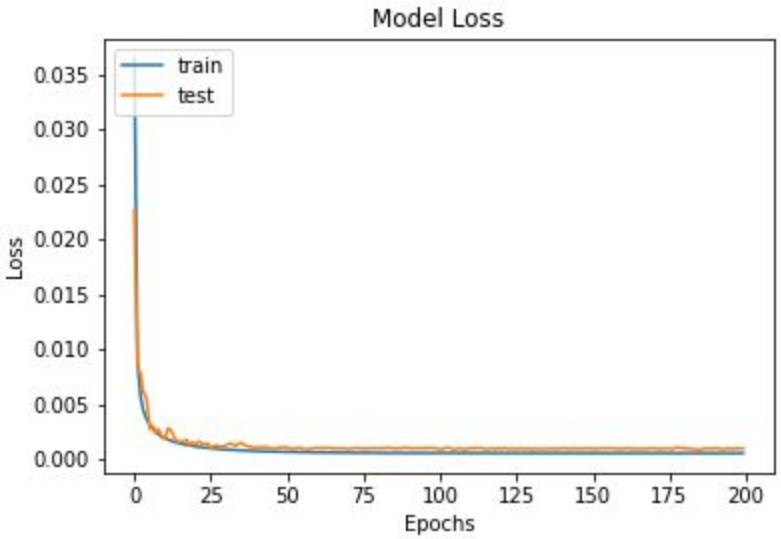
1. Convolutional Layers = 2
    a. Conv1: #filters = 32, kernel_size = 3
    b. MaxPooling Layer (2, 2)
    c. Conv2: #filters = 64, kernel_size = 3
    d. MaxPooling Layer (2, 2)
2. Epochs = 64
3. Batch Size = [25, 50, 100, 150, 200]



1. Epochs = 64
2. Batch Size = 200
3. Convolutional Layers = [2, 4, 6]
    a. Convolutional Layers = 2
        i. Conv1: #filters = 32, kernel_size = 3
        ii. MaxPooling Layer (2, 2)
        iii. Conv2: #filters = 64, kernel_size = 3
        iv. MaxPooling Layer (2, 2)
    b. Convolutional Layers = 4
        i. Conv1: #filters = 16, kernel_size = 3
        ii. Conv2: #filters = 32, kernel_size = 3
        iii. MaxPooling Layer (2, 2)
        iv. Conv3: #filters = 64, kernel_size = 3
        v. Conv4: #filters = 128, kernel_size = 3
        vi. MaxPooling Layer (2, 2)
    c. Convolutional Layers = 6
        i. Conv1: #filters = 8, kernel_size = 3
        ii. Conv2: #filters = 16, kernel_size = 3
        iii. Conv3: #filters = 32, kernel_size = 3
        iv. MaxPooling Layer (2, 2)
        v. Conv4: #filters = 64, kernel_size = 3
        vi. Conv5: #filters = 128, kernel_size = 3
        vii. Conv6: #filters = 256, kernel_size = 3
        viii. MaxPooling Layer (2, 2)



1. Batch Size = 200
2. Convolutional Layers = 3
    a. Conv1: #filters = 32, kernel_size = 5
    b. MaxPooling Layer (2, 2)
    c. Conv2: #filters = 64, kernel_size = 5
    d. MaxPooling Layer (2, 2)
    e. Conv3: #filters = 128, kernel_size = 5
3. Epochs = [8, 16, 32, 64, 128]



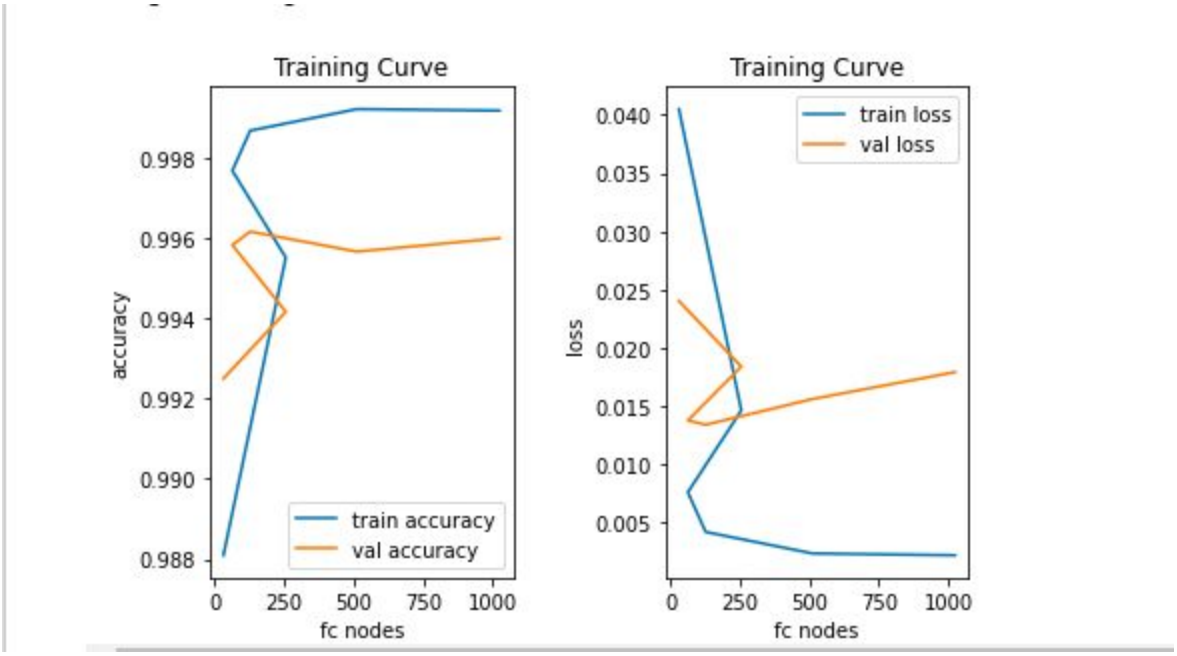
1. Epochs = 200
2. Batch Size = 64
3. Convolutional Layers = 5
    a. Conv1: #filters = 32, kernel_size = 3
    b. Conv2: #filters = 64, kernel_size = 3

c. MaxPooling Layer (2, 2)
d. Conv3: #filters = 128, kernel_size = 3
e. Conv4: #filters = 256, kernel_size = 3
f. MaxPooling Layer (2, 2)
g. Conv5: #filters = 256, kernel_size = 3

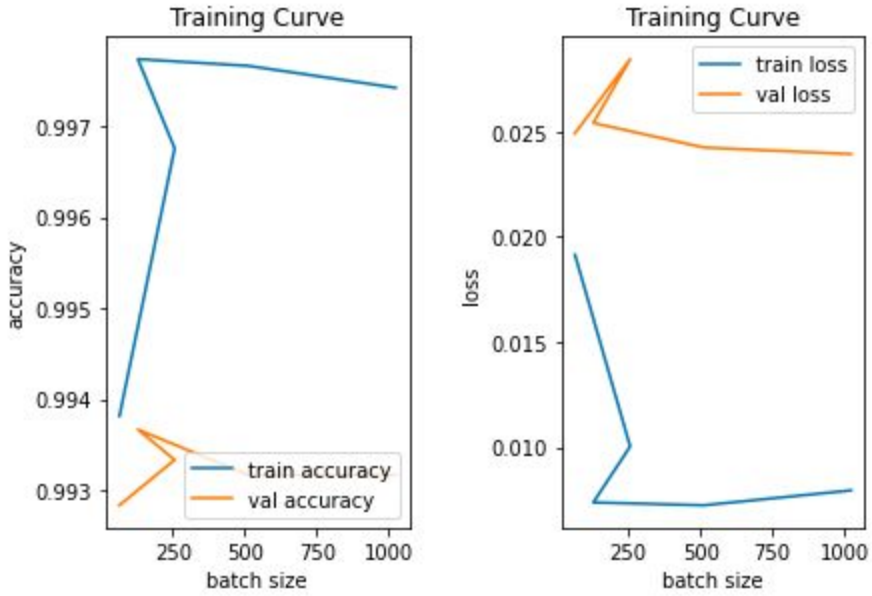
Classifier

Αρχιτεκτονική 1: Convolutional Layers = 2
Conv1 - #filters = 32, kernel_size = 5
Conv2 - #filters = 64, kernel_size = 5

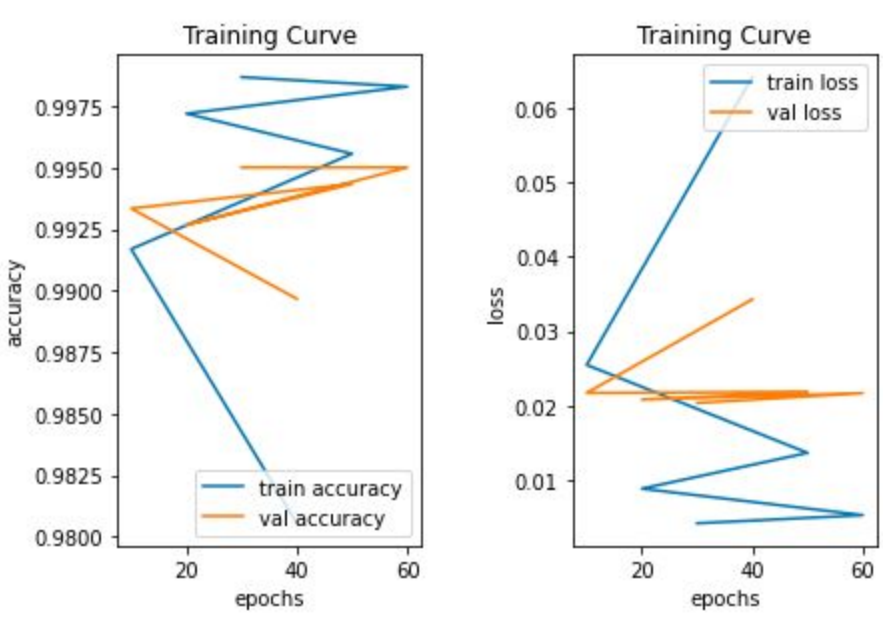
1. Epochs = 40
   batch size = 256
   nodes in fully-connected layer = [32, 64, 128, 256, 512, 1024]



2. Epochs = 40
   nodes in fully-connected layer = 128
   batch size = [32, 64, 128, 256, 512, 1024]



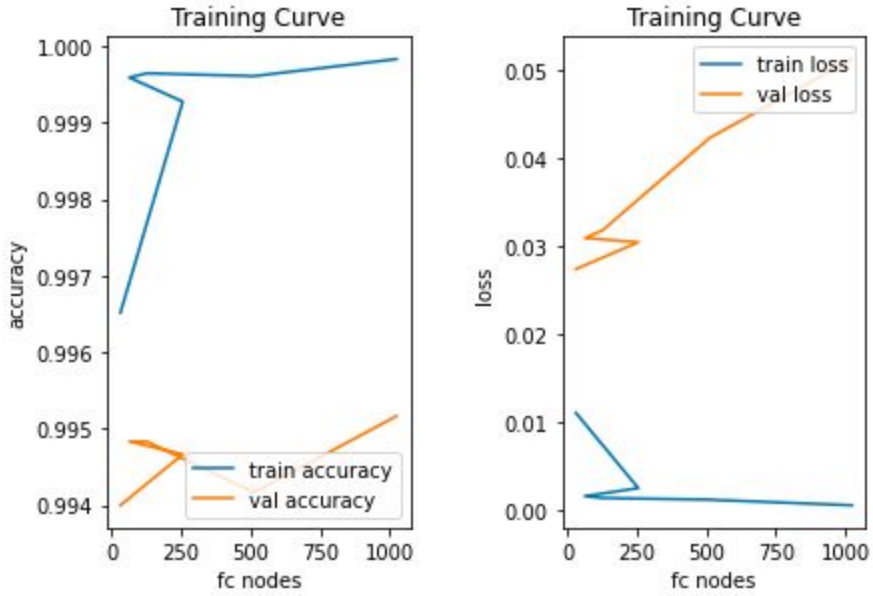
3. batch size = 256
   nodes in fully-connected layer = 128
   Epochs = [10, 20, 30, 40, 50, 60]



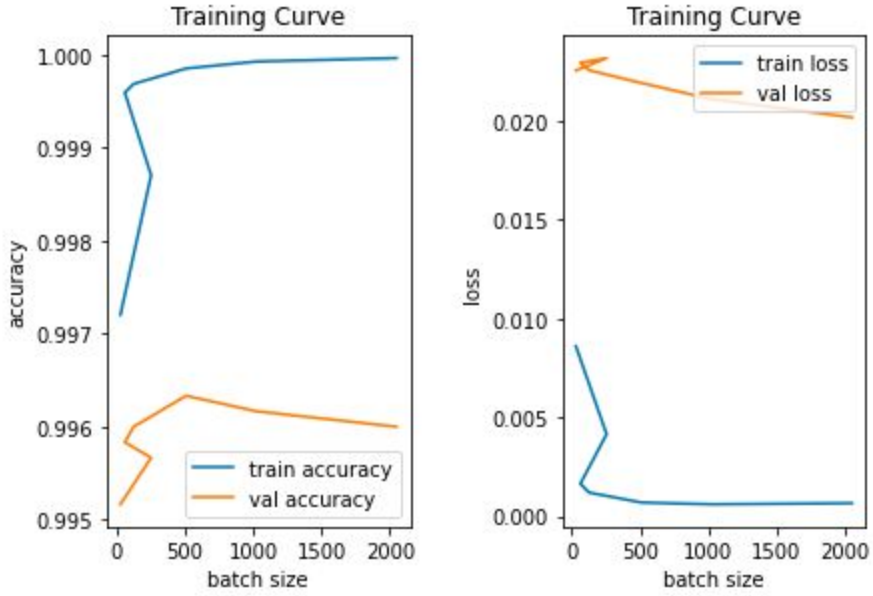
Αρχιτεκτονική 2: Convolutional Layers = 4
Conv1 - #filters = 32, kernel_size = 5
Conv2 - #filters = 32, kernel_size = 5
Conv3 - #filters = 64, kernel_size = 5

Conv4 - #filters = 64, kernel_size = 5

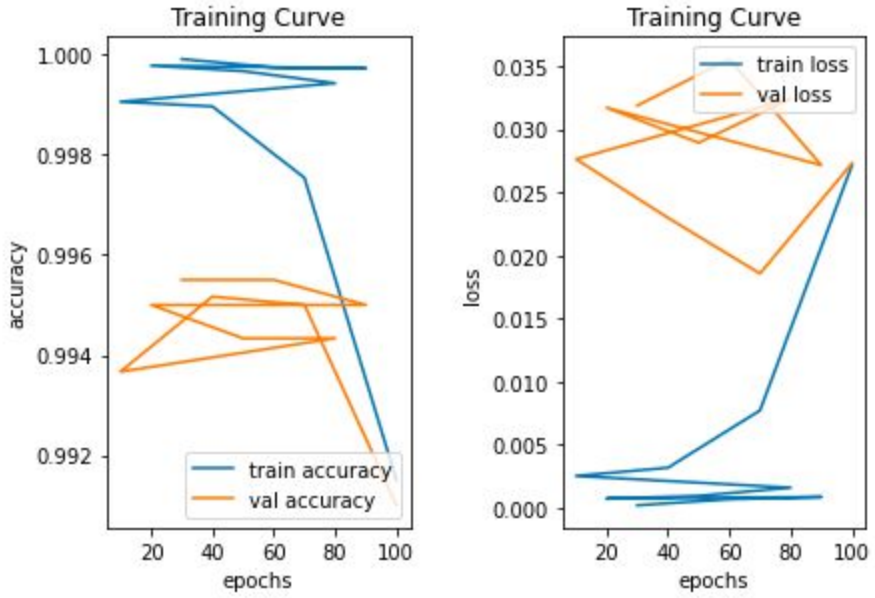
1. Epochs = 30
   batch size = 128
   nodes in fully-connected layer = [32, 64, 128, 256, 512, 1024]



2. Epochs = 30
   nodes in fully-connected layer = 128
   batch size = [32, 64, 128, 256, 512, 1024, 2048]



3. batch size = 128
   nodes in fully-connected layer = 128
   Epochs = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

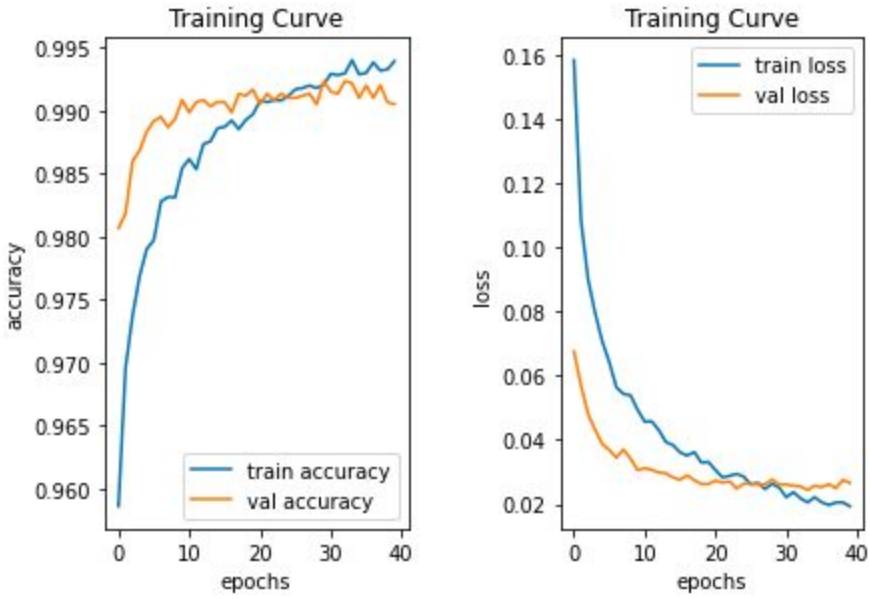


Από τα παραπάνω πειράματα (και για τις 2 αρχιτεκτονικές) διαπιστώνουμε τα εξής:

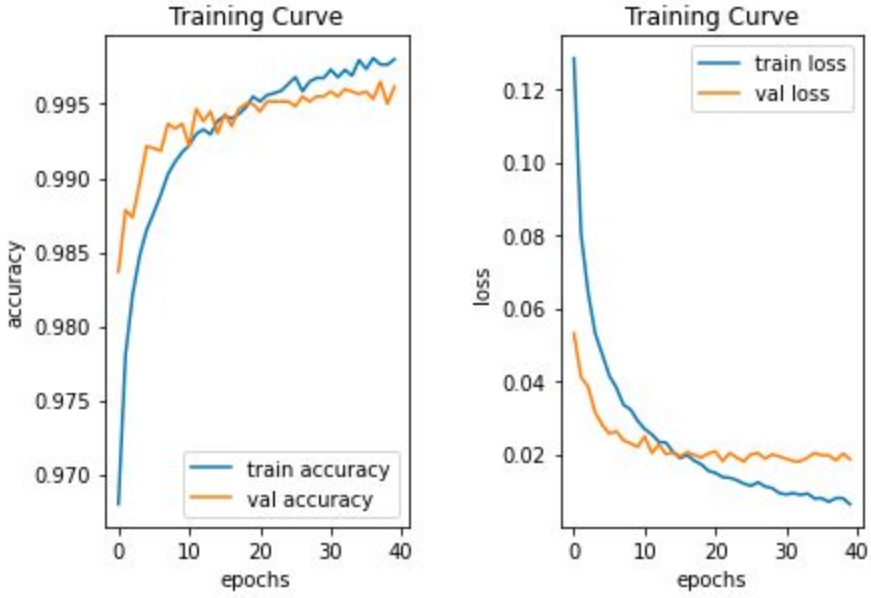
1. Η βέλτιστη τιμή του αριθμού των κόμβων του fully-connected layer φαίνεται να είναι 128 η 256, καθώς για αυτές τις τιμές το validation accuracy παίρνει την μέγιστη και το validation loss την ελάχιστη τιμή τους.
2. Σχετικά με το batch size, στην αρχιτεκτονική 1 (σχήμα 2) βλέπουμε ότι το validation accuracy παίρνει την μέγιστη τιμή μεταξύ 128 και 256 και στη συνέχεια μειώνεται. Από την άλλη πλευρά, το validation loss μειώνεται ελάχιστα όσο το batch size αυξάνεται. Στην αρχιτεκτονική 2 (σχήμα 2), το validation accuracy παίρνει τη μέγιστη τιμή για batch size μεταξύ 256 και 512, ενώ για το validation loss παρατηρείται το ίδιο φαινόμενο με την αρχιτεκτονική 1, δηλαδή μειώνεται ελάχιστα και με πολύ αργό ρυθμό όσο το batch size αυξάνεται. Απ' ότι φαίνεται τιμές του batch size μεγαλύτερες του 128-256 δουλεύουν καλύτερα με "deeper" architectures, δηλαδή με αρχιτεκτονικές που έχουν μεγάλο αριθμό hidden layers, και συνεπώς μεγάλο αριθμό από trainable parameters.
3. Αν και για τον αριθμό των epochs τα διαγράμματα δεν δείχνουν ξεκάθαρα ποια είναι η βέλτιστη τιμή ως προς το validation accuracy και το validation loss, από τα πολλά πειράματα που εκτελέσαμε, epochs = 30 η epochs = 40 δουλεύουν αρκετά καλά.

Οπότε, με βάση τα παραπάνω συμπεράσματα παραθέτουμε τις καλύτερες τιμές test loss και test accuracy που πήραμε για τις 2 αρχιτεκτονικές (**για ένα model μόνο**) :

1. Αρχιτεκτονική 1: fc_nodes = 128, batch size = 128, epochs = 40
   a. **Test Accuracy = 0.994**
   b. **Test Loss = 0.019**



2. Αρχιτεκτονική 2: fc_nodes = 128, batch size = 256 , epochs = 40
    a. **Test Accuracy = 0.996**
    b. **Test Loss = 0.013**



Ο αριθμός των epochs και το batch size παρέμειναν ίδια και στα 2 στάδια εκπαίδευσης του κάθε μοντέλου.