ΤΕΧΝΙΚΗ ΑΝΑΦΟΡΑ

`Για την Τέταρτη Εργασία στο μάθημα

«Όραση Υπολογιστών»

Καθηγητής: Ιωάννης Πρατικάκης

Στέλιος Μούσλεχ

AM:57382

19/1/2020,Ξάνθη

ΕΙΣΑΓΩΓΗ

Στην παρακάτω τεχνική αναφορά θα περγραφεί η λύση της τέταρτης εργασίας στο μάθημα «Όραση Υπολογιστών» αναλύοντας σε περιληπτικό επίπεδο το απαραίτητο θεωρητικό υπόβαθρο της επίλυσης καθώς θα γίνει και η ανάλυση μερικών ενδιάμεσων αποτελεσμάτων και προβλημάτων που αντιμετωπίστηκαν. Θα περιγραφθούν οι δύο υλοποιήσεις που καταχωρύθηκαν στο Kaggle.

Αναφορικά με την εργασία:

Η άσκηση αφορά στην υλοποίηση ενός συνελικτικού νευρωνικού δικτύου για την ταξινόμηση εικόνων.(6 κλάσεις εικόνων).Όπως σημειώθηκε στο τελευταίο εργαστήριο, μπορείτε να δημιουργήσετε πολλαπλά notebooks (διαφορετικές υλοποιήσεις) ΑΛΛΑ στο τέλος μπορείτε να υποβάλετε μόνο δύο από αυτές (βάσει της ακρίβειας στα δεδομένα ελέγχου που είναι διαθέσιμα). Υπενθυμίζεται ότι τουλάχιστον ένα (1), θα πρέπει να είναι μη-προ εκπαιδευμένο.

KAGGLE USERNAME: steliosmsl

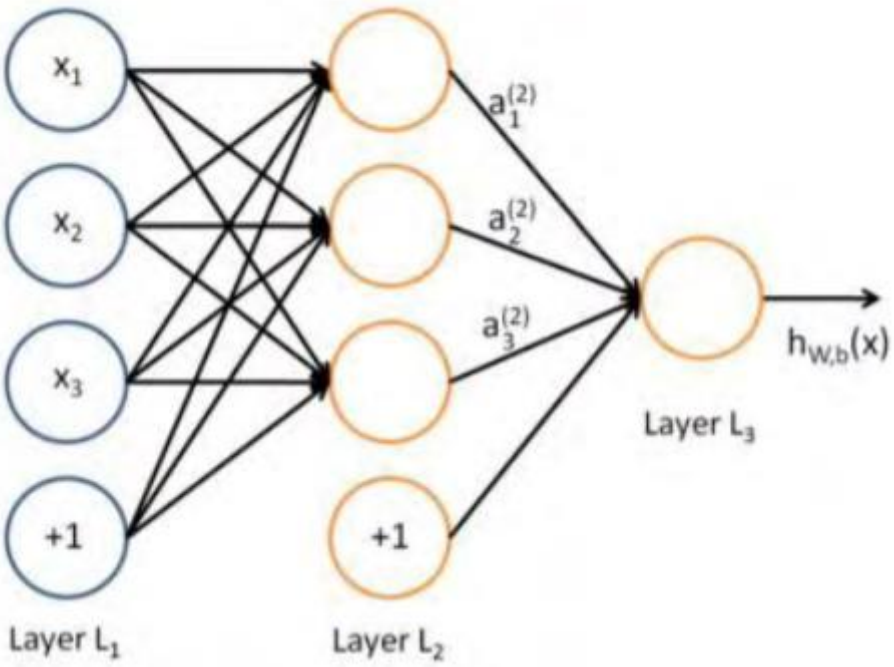
Display Name: Stelios Mouslech

Περιεχόμενα

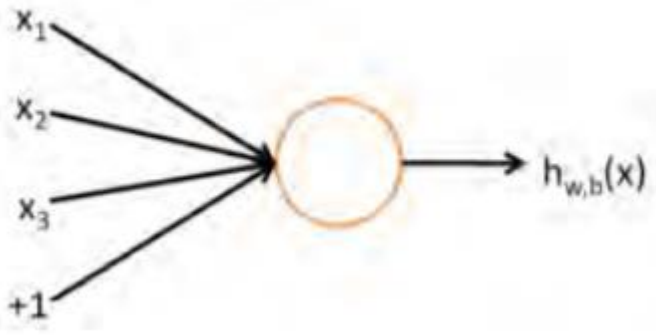
1.Συνοπτική Θεωρητική περιγραφή

1.1 Μοντέλο Νευρωνικών Δικτύων

Ένα νευρωνικό δίκτυο μπορεί να δημιουργηθεί συνενώνοντας πολλούς νευρώνες ή αλλιώς, έτσι ώστε η έξοδος του ενός να αποτελεί είσοδο σε κάποιον άλλο όπως φαίνεται στην εικόνα:



Για να περιγράψουμε τα νευρωνικά δίκτυα, θα ξεκινήσουμε από το απλούστερο δυνατό ΝΝ, αυτό που περιλαμβάνει έναν μόνο νευρώνα. Θα χρησιμοποιήσουμε το ακόλουθο σχήμα για να δείξουμε έναν νευρώνα:



Τα διάφορα σήματα Xj τα οποία αποτελούν είσοδο ενός νευρώνα i, πολλαπλασιάζονται με συντελεστές βαρύτητας wj,i. ν Η συνολική είσοδος στον νευρώνα i είναι τελικά το άθροισμα όλων των επιμέρους εισόδων της, μετά τον πολλαπλασιασμό τους με τους συντελεστές βαρύτητας.

Η έξοδος του τεχνητού νευρώνα προκύπτει από την εφαρμογή της συνάρτησης ενεργοποίησης (activation function) στην συνολική του είσοδο Xi . (Σημειώνουμε, ότι γενικά αντί τις μονάδας ( ) στο πιο πάνω άθροισμα, προτιμούμε να προσθέτουμε κάποια σταθερά , έτσι ώστε η τιμή της να μπορεί να μεταβάλλεται ανάλογα με τις ανάγκες του προβλήματος).

1.2 Μάθηση στα Νευρωνικά Δίκτυα

Η μάθηση στα Νευρωνικά δίκτυα συνίσταται στην αλλαγή των βαρών των συνδέσεων μεταξύ των νευρώνων. Οι Βασικές μέθοδοι μάθησης:

♦ Μάθηση με επίβλεψη (supervised learning)

♦Μάθηση χωρίς επίβλεψη (unsupervised learning)

Αναφορικά με την μάθηση με επίβλεψη: Η εκπαίδευση συνίσταται στον προσδιορισμό των κατάλληλων συντελεστών βάρους μεταξύ των συνάψεων και πραγματοποιείται με τη βοήθεια αλγορίθμων, κατάλληλων για την εκμάθηση του περιβάλλοντος και την βελτίωση της απόδοσής του. Η εκμάθηση ενός ΝΔ, γίνεται σταδιακά και σύμφωνα με καθορισμένους κανόνες. Μια επαναληπτική διαδικασία ρυθμίζει τα βάρη στις συνάψεις μεταξύ των επιπέδων κι έτσι το ΝΔ αποκτά περισσότερη “γνώση”.

Ένας αλγόριθμος για τον υπολογισμό και την ανανέωση των βαρών, είναι ο αλγόριθμος οπισθοδιάδοσης σφάλματος ή αλλιώς Backpropagation Algorithm.

Αρχικά για να μετρήσουμε την επίδοση του δικτύου μας στην εκπαίδευση του ορίζουμε μια συνάρτηση κόστους όπου ουσιαστικά μετράμε ποσοτικά το σφάλμα μεταξύ της προβλεπόμενης τιμής και της αληθινής τιμής των δεδομένων μας. Είναι προφανές λοιπόν ότι θέλουμε να την ελαχιστοποιήσουμε.

Το gradient descent αποτελεί τη κλασσική μέθοδο για να προσπαθήσουμε να ελαχιστοποιήσουμε την συνάρτηση κόστους μετακινούμενη πάντα στην κατεύθυνση με την πιο «απότομη κλίση προς τα κάτω» . Με βάση την λογική αυτή ανανεώνουμε τα βάρη μας στο δίκτυο μας.

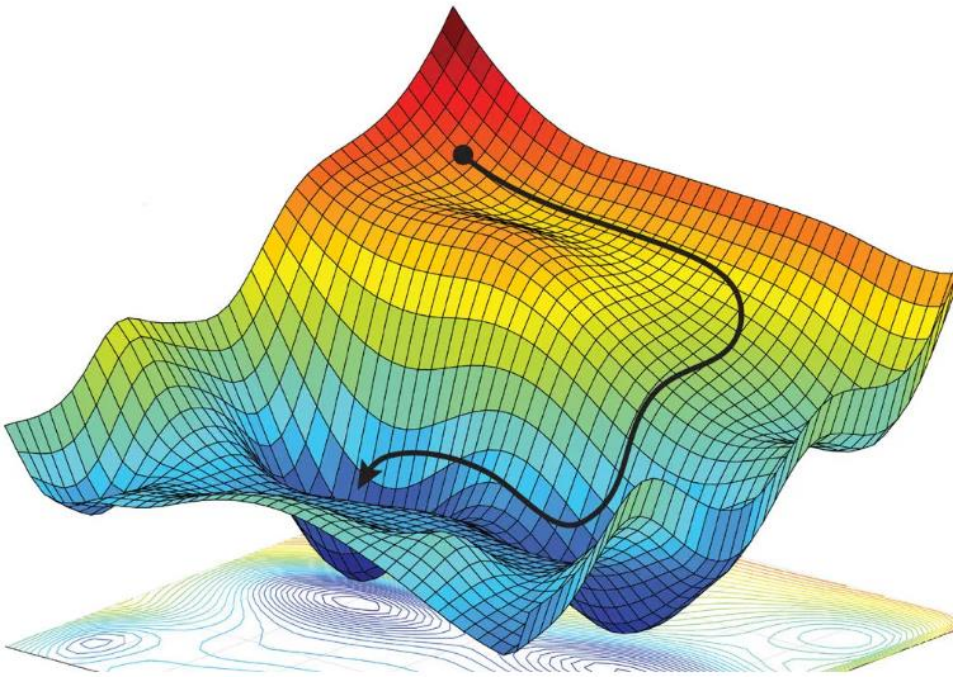
1.2.1 Epochs - Batches

Ο αλγόριθμος είναι επαναληπτικός δηλαδή για να έχουμε βέλτιστο αποτέλεσμα πρέπει να γίνει επαναληπτικά πολλές φορές. Οπότε ορίζουμε ως

- Μια Εποχή (Epoch) έχουμε όταν όλα τα training δεδομένα έχουν περάσει από το δίκτυο μας και προς τα μπροστά και προς τα πίσω

- Ένα Batch (ομάδα, παρτίδα) δεδομένων αποτελεί ένα μέρος των δεδομένων που εισάγονται μαζί στο νευρωνικό δίκτυο. Ουσιαστικά τα βάρη αναβαθμίζονται μετά από κάθε Batch

Τα παραπάνω αποτελούν υπερπαραμέτρους της διαδικασίας μάθηση και η σωστή επιλογή τιμών για αυτά γίνεται μέσω δοκιμής και σφάλματος .



Η μέθοδος της μάθησης αυτής δεν συγκλίνει πάντα στη βέλτιστη λύση. Υπάρχει περίπτωση να παγιδευτεί σε τοπικά ελάχιστα και έτσι να μην βρει τα βέλτιστα βάρη.

1.2.2 Optimizers – Accuracy

Όπως αναφέραμε πιο πάνω ο αλγόριθμος είναι επαναληπτικός και αναβαθμίζει τα βάρη του δικτύου μας ανάλογα με τη ποσότητα του cost function. Ο τρόπος που θα αλλάξουμε τα βάρη με σκοπό την ελαχιστοποίηση της cost function μας καθορίζεται από τον optimizer μας.

Μερικοί γνωστοί βασισμένοι στο Gradient Descent που αναφέραμε παραπάνω:

- Stochastic Gradient Descent
- Adagrad
- RMSprop
- Adam

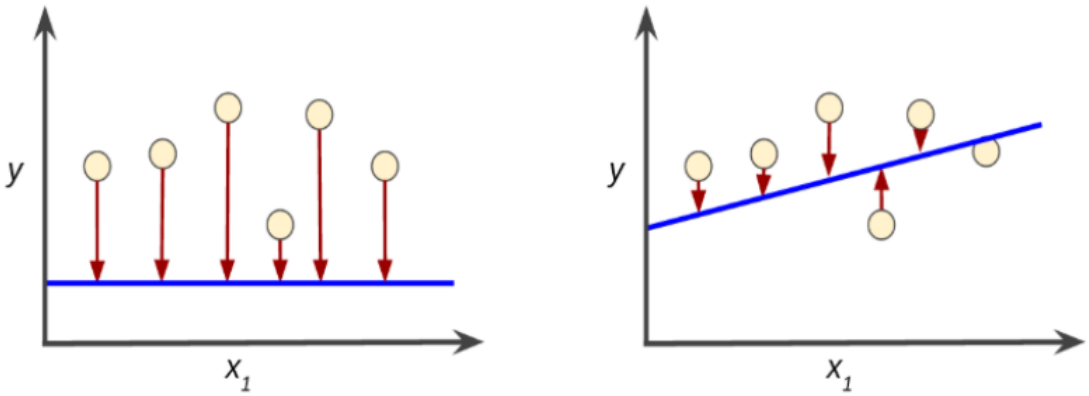
Εδώ αξίζει να αναφέρουμε και την πιο βασική παράμετρο των optimizers το Learning Rate. Ουσιαστικά πρέπει να καθορίσουμε πόσο μεγάλες αλλαγές θα κάνουμε στα

βάρη μας («πόσο θα κατέβουμε στο cost function ουσιαστικά»). Το βήμα με το οποίο γίνονται αυτές οι αλλαγές αποτελεί το Learning Rate. Αυτό αποτελεί μια πολύ μιρκή τιμής με την οποία πολλαπλασιάζουμε τα gradients για να επιτρέψουμε μικρές αλλαγές στα βάρη μας.

Αν το learning rate είναι πολύ μεγάλο τότε μπορεί να μην καταλήξουμε σε βέλτιστη λύση , αν από την άλλη είναι πολύ μικρό μπορεί να παγιδευτούμε σε τοπικό ελάχιστο που δεν είναι το βέλτιστο.

Έχοντας ορίσει τα παραπάνω είναι χρήσιμο να ορίσουμε και κάποια άλλα μετρικά που θα δούμε και πρακτικά στις υλοποιήσεις μας.

- Accuracy: Αποτελεί το ποσοστό των επιτυχών προβλέψεων σε κάποιο dataset(training,validation,test περισσότερα για αυτά παρακάτω).
- Loss: Μας δείχνει κατά μέσο όρο πόσο απόκλιση είχαμε στις τιμές των προβλέψεων από τα πραγματικές τιμές του dataset. (Μπορούμε να το σκεφτούμε σαν πόσο “confident” είναι το μοντέλο μας στις προβλέψεις μας



- Τα κόκκινα βέλη αναπαριστούν το loss
- Οι μπλε γραμμές είναι οι προβλέψεις μας

Είναι προφανές ότι το αριστερό διάγραμμα έχει μεγαλύτερο loss από το δεξιά. Περισσότερα για το πως να ερμηνεύουμε διαφορές σε Training Accuracy, Validation Accuracy, Training Loss και Validation Loss θα συζητηθούν παρακάτω.

1.3Αρχιτεκτονική των CNN

Τα Συνελικτικά Νευρωνικά Δίκτυα (ΣΝΔ) ή στα Αγγλικά Convolutional Neural Networks (CNN), χωρίζονται σε δυο μεγάλες κατηγορίες τα Αβαθή Νευρωνικά Δίκτυα (Shallow Neural Networks) και τα Βαθιά Νευρωνικά Δίκτυα (Deep Neural Networks). Ένα συνελικτικό επίπεδο (convolutional layer) είναι ουσιαστικά ένα σύνολο από νευρώνες που εκτελούν συνέλιξη των φίλτρων που έχουν προκαθοριστεί, με την εικόνα-διάνυσμα που δέχονται στην είσοδο. Κάθε επίπεδο μπορεί να περιλαμβάνει νευρώνες που εκτελούν συνέλιξη, διαδικασίες pooling, εισαγωγή μη γραμμικότητας ή ακόμη και κανονικοποίηση, ενώ έχει διακριτές εισόδους και εξόδους. Οι διαστάσεις των φίλτρων που περιλαμβάνουν, ο αριθμός τους και το βάθος τους (αριθμός καναλιών) μπορεί να διαφέρει σημαντικά ανάλογα με το πρόβλημα.

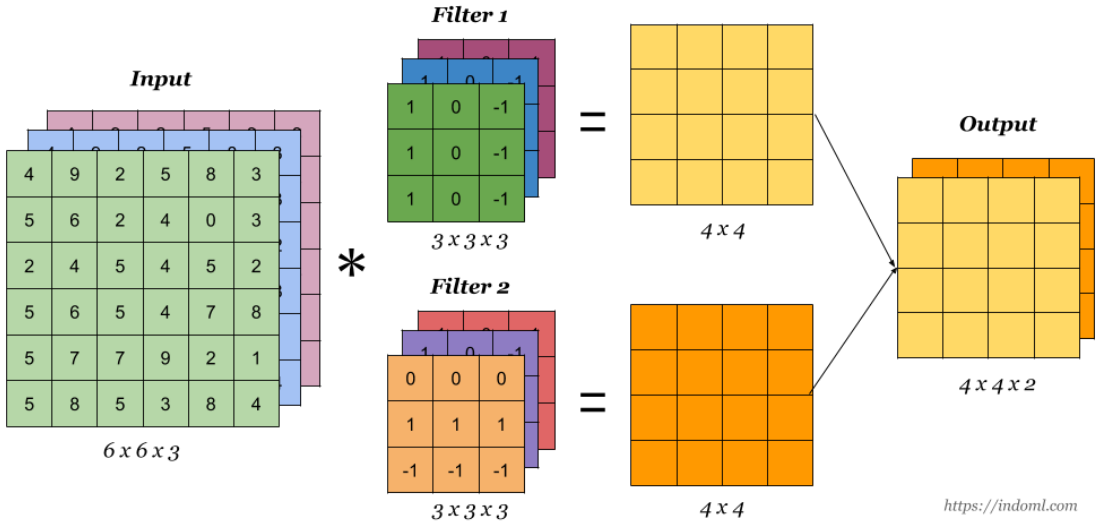
Τα layers ενός CNN αποτελούνται συνήθως από

1. Convolutional Layers

2. Pooling Layers

3. Fully Connected Layers

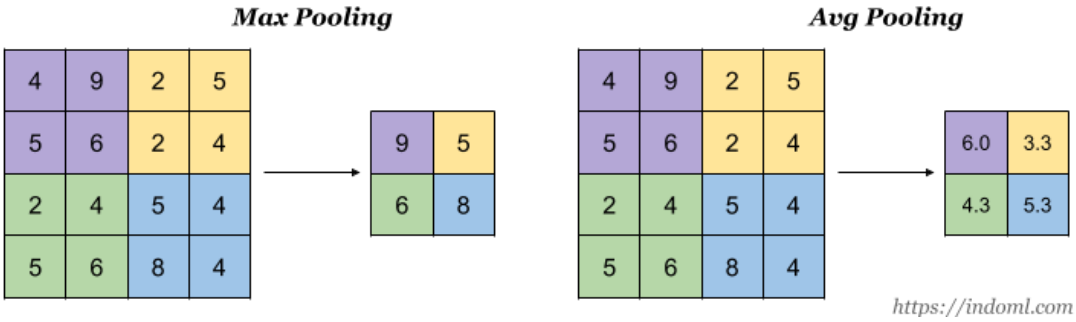
Η κεντρική ιδέα στην οποία βασίζονται τα CNN, είναι ότι τα χαρακτηριστικά ενός patch εικόνας, είτε αυτά αφορούν κλίσεις, είτε ακμές κλπ, είναι τοπικά εντοπισμένα. Στα layers που γίνονται convolutions υπολογίζουμε ουσιαστικά χαρακτηριστικά με διάφορα φίλτρα με διάφορες τιμές.



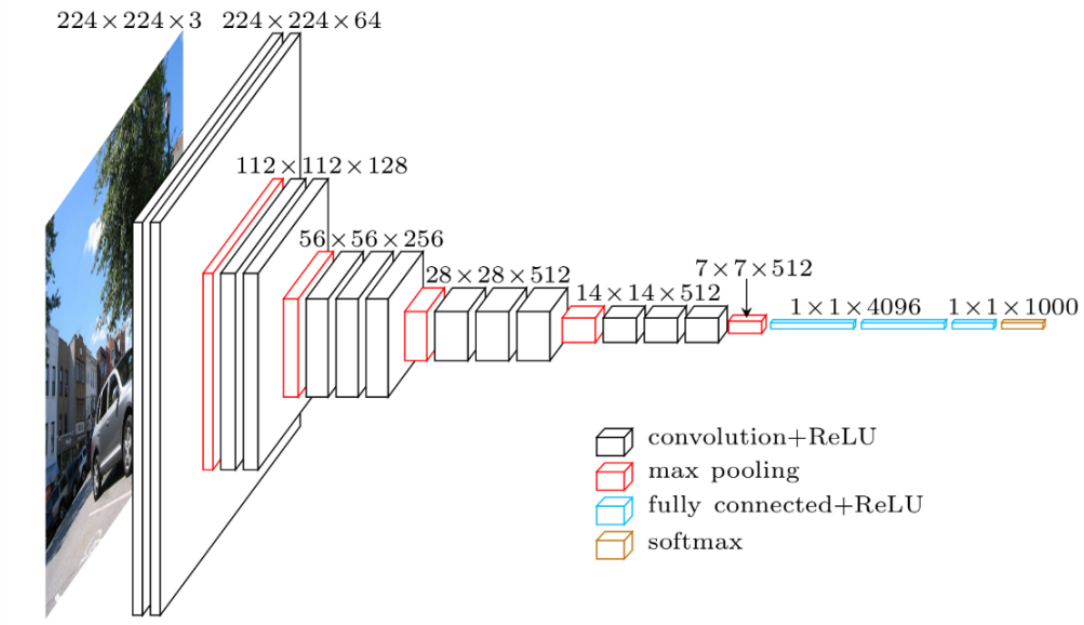
Τα επίπεδα συγκέντρωσης (pooling layers) στα CNNs, συνοψίζουν τις εξόδους γειτονικών γκρουπ νευρώνων εντός ενός παραθύρου (patch) με μια αντιπροσωπευτική

τιμή, ενώ συνήθως τα γειτονικά παράθυρα δεν επικαλύπτονται. Πρόκειται ουσιαστικά για μια διαδικασία υπο-δειγματοληψίας των δεδομένων.

Το pooling αποτελεί μια πολύ βασική λειτουργία για κάθε CNN, αφού απλοποιεί πολύ τη διαδικασία λόγω της σημαντικής μείωσης των δεδομένων κι επομένως του αριθμού των απαιτούμενων πράξεων. Οι επικρατέστερες κατηγορίες του pooling είναι το max, sum και average pooling, ενώ μπορεί τα παράθυρα που χρησιμοποιούνται να επικαλύπτονται ή και όχι ανάλογα με τις ανάγκες του προβλήματος. Η διαδικασία του pooling, εκτός απ τη μείωση του μεγέθους των δεδομένων, μας δίνει τη δυνατότητα προσθήκης περισσότερης πληροφορίας στην αρχική εικόνα μέσω των αρχικών διαστάσεων ενώ είναι ανεξάρτητο μικρών μετασχηματισμών.



Τέλος για να κάνουμε το classification θα χρησιμοποιήσουμε ένα Fully connected network στο τέλος του CNN μας για να μάθουμε τους μη γραμμικούς συνδυασμούς των χαρακτηριστικών που προέκυψαν από τα προηγούμενα επίπεδα και έτσι να κάνουμε την ταξινόμηση.



2.Περιγραφή Των Υλοποιήσεων μου.

2.1. Ανάλυση δεδομένων

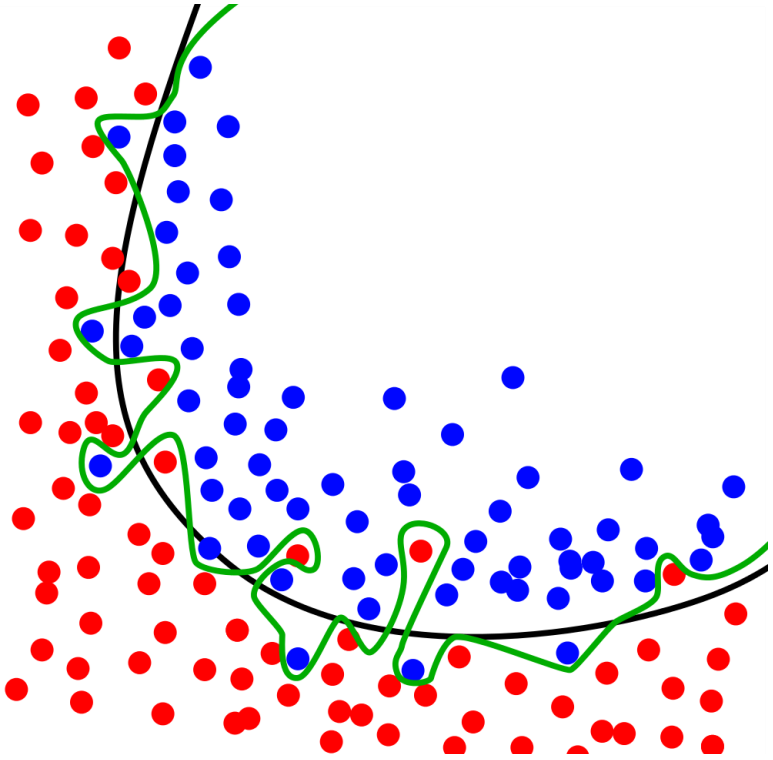
Παρατηρώντας τα δεδομένα μας βλέπουμε ότι για ένα πρόβλημα με 10 κλάσεις με περίπου 400 εικόνες ανά κλάση δεν έχουμε πολύ μεγάλο Dataset. Θα πρέπει να είμαστε προσεκτικοί να μην καταλήξουμε σε overfitting(περισσότερα παρακάτω) .

Τα δεδομένα έχουν ήδη χωριστεί σε training,validation και test data. Για την εκπαίδευση θα χρησιμοποιήσουμε τα training και validation δεδομένα μας (που είναι ήδη σε υποφακέλους για κάθε κλάση και αναγνωρίζονται έτσι από την ImageDataGenerator συνάρτηση).

- Τα Training δεδομένα είναι τα δεδομένα στα οποία το δίκτυο εκπαιδεύεται δηλαδή «μαθαίνει» τα βάρη του
- Τα validation δεδομένα είναι αυτά στα οποία το εκπαιδευμένο δίκτυο δοκιμάζει την ακρίβεια του (δεν έχει εκπαιδευτεί σε αυτά τα δεδομένα) και χρησιμοποιείται για να ρυθμίσουμε τις Υπερπαραμέτρους του δικτύου μας και να ελέγχουμε καλύτερα την πραγματική του ακρίβεια και να αντιληφθούμε πότε το μοντέλο μας κάνει overfit
- Τα test δεδομένα είναι αυτά στα οποία μετά το τελικό πέρας της εκπαίδευσης δοκιμάζουμε το δίκτυο μας για να το αξιολογήσουμε.

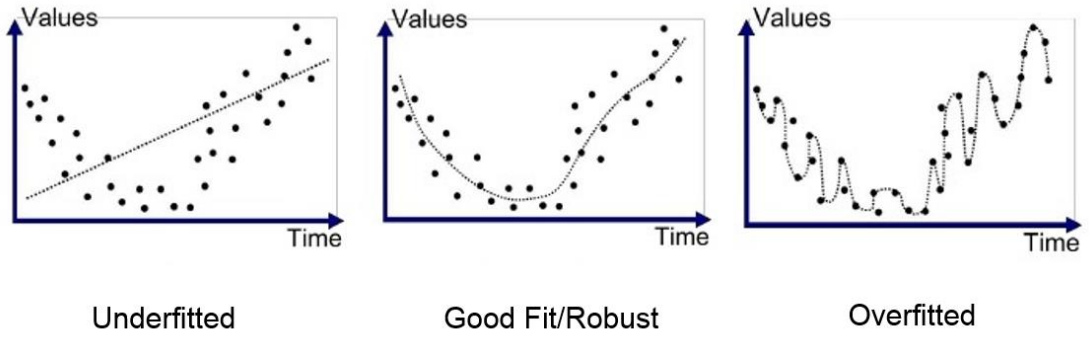
2.2 Overfitting - Underfitting

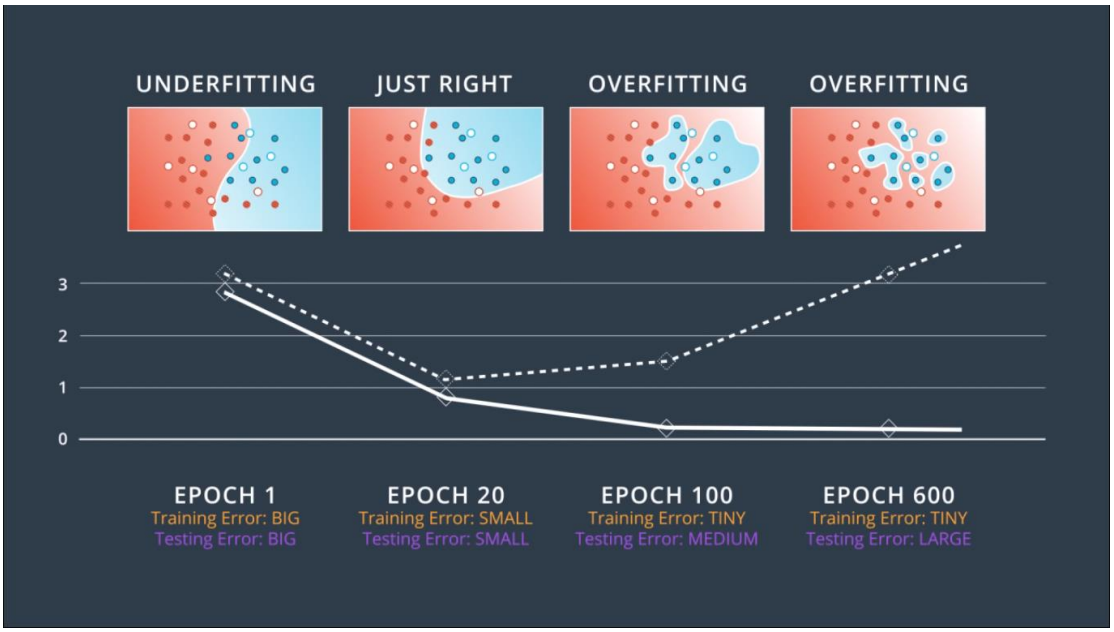
Overfit ουσιαστικά συμβαίνει στο μοντέλο μας όταν «μαθαίνει» πολύ καλά τα training δεδομένα και χάνει την ικανότητα να γενικεύει με αποτέλεσμα να αποδίδει άσχημα σε καινούρια δεδομένα στα οποία δεν εκπαιδεύτηκε. Ουσιαστικά έχει μάθει και τον «Θόρυβο» των δεδομένων εκπαίδευσης ως μέρος των χαρακτηριστικών.



Πράσινη Γραμμή: Overfit Μαύρη Γραμμή: Καλό Fit

Underfitting συμβαίνει στο μοντέλο μας όταν ουσιαστικά δεν μπορεί να αντιληφθεί κάποιες κοινές «τάσεις» των δεδομένων μας. Αυτό μπορεί να συμβεί είτε γιατί δεν εκπαιδεύουμε αρκετά μεγάλο μοντέλο σε σύγκριση με την ανάγκη του προβλήματός μας είτε γιατί δεν εκπαιδεύουμε για αρκετές εποχές το μοντέλο μας





2.3 Πρώτη υλοποίηση – Δικό μου δίκτυο

Στην πρώτη από τις submitted υλοποιήσεις μου αρχικά έβλεπα μεγάλη διαφορά μεταξύ training (πολύ μικρό loss και μεγάλο accuracy) με validation (πολύ μεγάλο συγκριτικά loss και μικρότερο accuracy). Οπότε το μοντέλο μου έκανε overfit στην προσπάθεια αυτή έκανα μερικές αλλαγές στο μοντέλο μο στην προσπάθεια να μειώσω το overfit αυτά είναι:

- Data Augmentation
- Callbacks
- Dropout

Πιο συγκεκριμένα αναφορικά με το Data augmentation Θέλαμε τα δεδομένα που δίνει στο δίκτυο ο ImageDataGenerator να είναι μετασχηματισμένα ώστε το μοντέλο μας να μπορέσει να μάθει περισσότερο γενικά χαρακτηριστικά των δεδομένων και να μπορεί να γενικεύει καλύτερα. Τέτοιο μετασχηματισμοί μπορεί να είναι η περιστροφή της εικόνας το zooming το fllping κ.α

2.3.1 Data Augmentation

Στον κώδικά μας αυτά μπαίνουν σαν παραμέτρους στο ImageDataGenerator (**Εδώ να τονίσουμε ότι τα «πειραγμένα» δεδόμενα δεν προστίθονται στα ήδη υπάρχοντα αλλά ουσιαστικά τα αντικαταστούν δηλαδή ο αριθμός ο συνολικός μένει ίδιος**)

Στον υλοποίηση μου το data augmentation έγινε εδώ

```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=30,
zoom_range=0.15,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.15,
horizontal_flip=True,
fill_mode="nearest")
```

Επίσης να σημειώσουμε ότι δεν έχει νόημα να κάνουμε augment τα validation δεδομένα μας αφού θα χαθεί έτσι το νόημα που το κάνουμε.

2.3.2 Callbacks

Τα callbacks γενικά αποτελούν ένα σύνολο συναρτήσεων που μας επιτρέπουν να έχουμε παραπάνω έλεγχο και μετρικά κατά την διάρκεια της εκπαίδευσης.Για να φροντίσουμε να μην κάνουμε overfit το μοντέλο μας χρησιμοποίησα δύο συναρτήσεις:

- ModelCheckpoint
  Αυτή η συνάρτηση φροντίζει να κάνει save το μοντέλο μας σε συγκεκριμένες εποχές κατά το training.Πιο συγκεκριμένα μπορούμε να κάνουμε save για παράδειγμα μόνο την εποχή όπου το training accuracy είναι μέγιστοΣτην δικιά μας περίπτωση μια καλή επιλογή είναι να κάνουμε save μόνο την εποχή όπου το val_loss είναι ελάχιστο. Αν έχουμε διαλέξει και πολλές εποχές θα είναι ένα καλό μοντέλο . Στον Κώδικα

  ```
  call = callbacks.ModelCheckpoint("best.h5", monitor='val_loss', mode = 'min', verbose=1, save_b
  est_only = True)
  ```

- EarlyStopping

  Επίσης για να αποτρέψουμε το μοντέλο μας από το να κάνει overfit μπορούμε όταν το validation loss αρχίζει να ανεβαίνει για αρκετές εποχές να σταματάμε το training ώστε να μην εκπαιδεύουμε για παραπάνω εποχές.

```
early = callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=1, mode='auto')
```

Και τα δύο callbacks τα καλούμε όταν κάνουμε fit το μοντέλο μας για να κάνει train στην παράμετρο callbacks :

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=train_generator.samples/train_generator.batch_size ,
      epochs=60,
      validation_data=validation_generator,
      validation_steps=validation_generator.samples/validation_generator.batch_size,
      verbose=1,callbacks=[call,early])
```

2.3.3 Dropouts

Dropout ονομάζουμε την «αφαίρεση» μερικών νευρώνων (μαζί με τις εισόδους εξόδους τους) στο δίκτυο μας κατά την διάρκεια της εκπαίδευσης. Ο λόγος που το κάνουμε αυτό είναι για να προστατευτούμε από το να κάνει το μοντέλο μας overfit

Αυτό το κάνει καθώς ένας νευρώνας μπορεί να γίνει πολύ εξαρτημένος από συγκεκριμένες εισόδους του. Αν αυτοί οι είσοδοι δεν είναι πάντα στην εκμάθηση τότε ο νευρώνας μαθαίνει να χρησιμοποιεί όλες τις εισόδους του οδηγώντας σε καλύτερο Generalization

Στον μοντέλο μας συνήθως τα βάζουμε στα fully connected layers (όχι πάντα) και στον κώδικα μου γίνεται στην εντολή:

```
model.add(layers.Dropout(0.5))
```

Η παράμετρος αποτελεί το rate των νευρώνων που θα αφαιρούνται κάθε φορά.

2.3.4 Αποτελέσματα

Δοκιμάζοντας διάφορες παραμέτρους (Μεγάλα και μικρά Batch Sizes, Πολλές παραπάνω εποχές και διαφορικές αρχιτεκτονικές) τα ποσοστά μου στο test submission σετ κυμαίνονταν μεταξύ 50-60 %. Εν τέλει διάλεξα να κάνω submit αυτό με το μεγαλύτερο test accuracy σε συνδυασμό με μικρότερο σχετικά Validation Loss .

Να σημειωθεί ότι καλά αποτελέσματα αλλά σχετικά κοντά είχα με batch size 32 και 64 αν πήγαινα πιο πάνω άρχισαν να χειροτερεύουν

Με καλύτερο μοντέλο :

```
Epoch 39/60
39/38 [==============================] - 23s 578ms/step - loss: 0.9684 - acc: 0.6299 - val_l
oss: 0.8124 - val_acc: 0.6559

Epoch 00039: val_loss improved from 0.82059 to 0.81243, saving model to best.h5
```

Το Notebook αυτό βρίσκεται στον φάκελο ως : FirstMynet.ipynb

```
model =models.Sequential()

model.add(layers.Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (128,128,3)))
model.add(layers.Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))


model.add(layers.Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(layers.Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2), strides=(2,2)))


model.add(layers.Flatten())
model.add(layers.Dense(256, activation = "relu"))
model.add(layers.Dropout(0.5))

model.add(layers.Dense(6, activation = "softmax"))
```

2.4 Δεύτερη υλοποίηση – Transfer Learning – Fine Tuning

2.4.1 Περιγραφή Transfer Learning – Υλοποίησης

Το αποτέλεσμα με το μοντέλο το παραπάνω δεν ήταν πολύ καλό έτσι αποφάσισα να κάνω μια διαφορετική προσέγγιση που είναι και πιο καλή σε μικρά dataset

Μια διαφορετική προσέγγιση για ένα τέτοιο πρόβλημα είναι να κάνουμε Transfer Learning-Fine Tuning από μια ήδη υπάρχουσα αρχιτεκτονική δικτύου προ εκπαιδευμένη σε ένα παρόμοιο πρόβλημα (συνήθως) και στη συνέχεια να βάλουμε τα δικά μας Fully Connected Layers για το classification στο πρόβλημα μας και είτε να έχουμε «παγωμένα» τα Layers μας και να εκπαιδεύονται μόνο τα fully connected που προσθέσαμε ή να εκπαιδεύουμε τα ήδη υπάρχοντα βάρη (καλύτερα με μικρό learning rate) .

Στην δικά μου υλοποίηση φόρτωσα τον VGG16 με τα βάρη του από το ImageNet και δοκιμάζοντας είχα καλύτερα αποτελέσματα με όλα τα layers να μπορούν να εκπαιδευτούν και με ένα μικρό learning rate (+ Data Augmentation,callbacks)

Βλέπουμε πως παίρνουμε το Pre-Trained μοντέλο

```
# Create the model
model=models.Sequential()

VGG= VGG16(include_top=False,input_shape=(128,128,3),weights='imagenet')

model.add(VGG)
model.add(layers.Flatten())
model.add(layers.Dense(6, activation = "softmax"))

# Show a summary of the model. Check the number of trainable parameters
model.summary()
```

Εδώ βλέπουμε και πως παγώνουμε κάποια layers για να μην εκπαιδευτούν αν και δεν προτιμήθηκε στο τέλος

```
from keras.applications import VGG16
#Load the VGG model
vgg_conv = VGG16(weights='imagenet', include_top=False, input_shape=(image_size, image_size, 3))
for layer in vgg_conv.layers:
    layer.trainable = False
model.add(vgg_conv)

# Add new layers
model.add(layers.Flatten())
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(3, activation='softmax'))
```

2.4.2 Αποτελέσματα

Με untrainable όλα τα layers του VGG δεν είχα καλά αποτελέσματα (μέγιστο 63% στα submissions) Αντίθετα με μικρό learning Rate και όλα τα layers εκπαιδεύσιμα έφτασα σε τεστ submissions στα 81%. Παρόλα αυτά διάλεξα τις παραμέτρους από ένα submission με 76% σε test submissions καθώς εκεί είχα πολύ μικρότερο val_loss σχετικά ίδιο val_acc και πιστεύω πως συνολικά σε όλο το test dataset θα έχει καλύτερα αποτελέσματα

```
Epoch 39/50
78/77 [==============================] - 25s 324ms/step - loss: 0.0205 - acc: 0.9960 - val_l
oss: 0.0018 - val_acc: 0.8842

Epoch 00039: val_loss improved from 0.07235 to 0.00179, saving model to best.h5
```

Το αρχείο με αυτή την υλοποίηση είναι ονομασμένο στο φάκελο ως:

TransferLearning.ipynb

3.ΠΗΓΕΣ-ΑΝΑΦΟΡΕΣ

[1] https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

[2] https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76

[3] https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9

[4] https://towardsdatascience.com/overfitting-vs-underfitting-ddc80c2fc00d

[5] https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

[6] https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

[7] https://algorithmia.com/blog/introduction-to-optimizers

[8]https://nemertes.lis.upatras.gr/jspui/bitstream/10889/9623/3/PapadopoulosAth%28phys%29.pdf