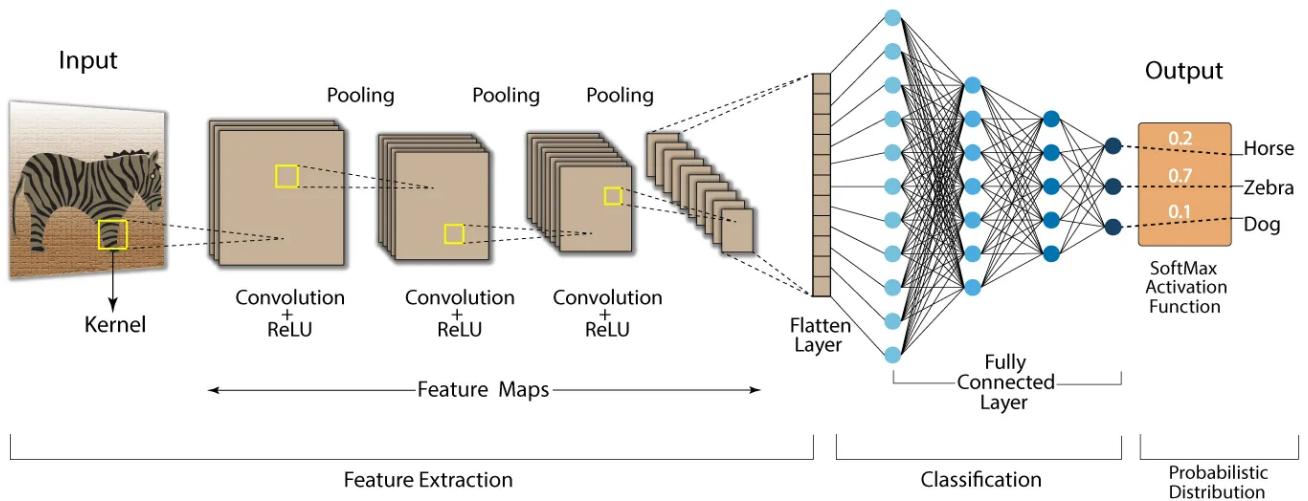


Αναγνώριση Προτύπων

2021-2022

Τελική Εργασία

Convolution Neural Network (CNN)



Καρυπίδης Ευστάθιος 57556

15/1/22, Ξάνθη

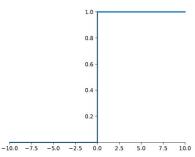
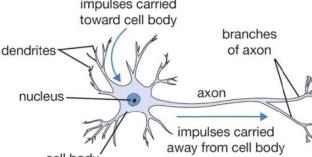
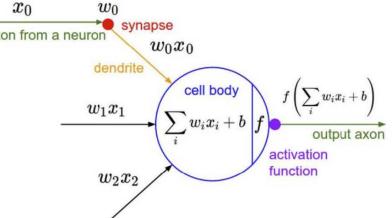
Στη συγκεκριμένη εργασία θα μελετηθούν τα παρακάτω Dataset:

1. Fashion Mnist
2. Cifar-10
3. Cifar-100

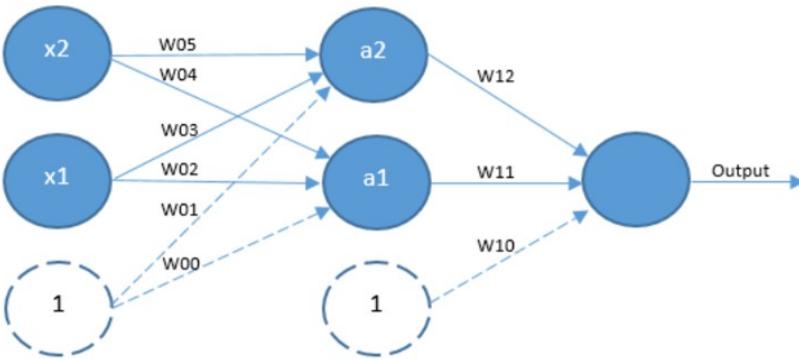
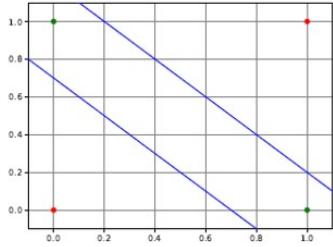
Αρχικά θα γίνει μία θεωρητική εισαγωγή. Στη συνέχεια θα παρουσιαστούν οι ασκήσεις που ζητάνε συγκεκριμένες αρχιτεκτονικές δικτύων και τέλος θα παρουσιαστούν τα δίκτυα δικής μου έμπνευσης και οι δοκιμές που έγιναν.

Εισαγωγή

Την τελευταία δεκαετία και ειδικότερα μετά το 2011-2012 τα νευρωνικά δίκτυα(Neural Networks) και η βαθιά μάθηση (Deep learning) οδήγησαν σε τεράστια άλματα σε πληθώρα πεδίων και εφαρμογών, μεταξύ άλλων και της υπολογιστικής όρασης. Σε αυτό το σημείο κρίνεται απαραίτητη μία σύντομη εισαγωγή. Παρά την τεράστια πρόοδο τα τελευταία χρόνια, οι βασικές ιδέες του deep learning που αποτελεί εξέλιξη των νευρωνικών δικτύων εισήχθησαν πολύ παλαιότερα. Ειδικότερα το 1943 οι Walter Pitts και Warren McCulloch περιέγραψαν την thresholded logic unit, μία μονάδα η οποία σχεδιάστηκε ώστε να μιμείται τον τρόπο με τον οποίο πίστευαν πως δουλεύουν οι νευρώνες. Η αναλογία μεταξύ βιολογικού και τεχνιτού νευρώνα φαίνεται παρακάτω.

Threshold Function	Biological - Artificial Neuron
$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$ 	 

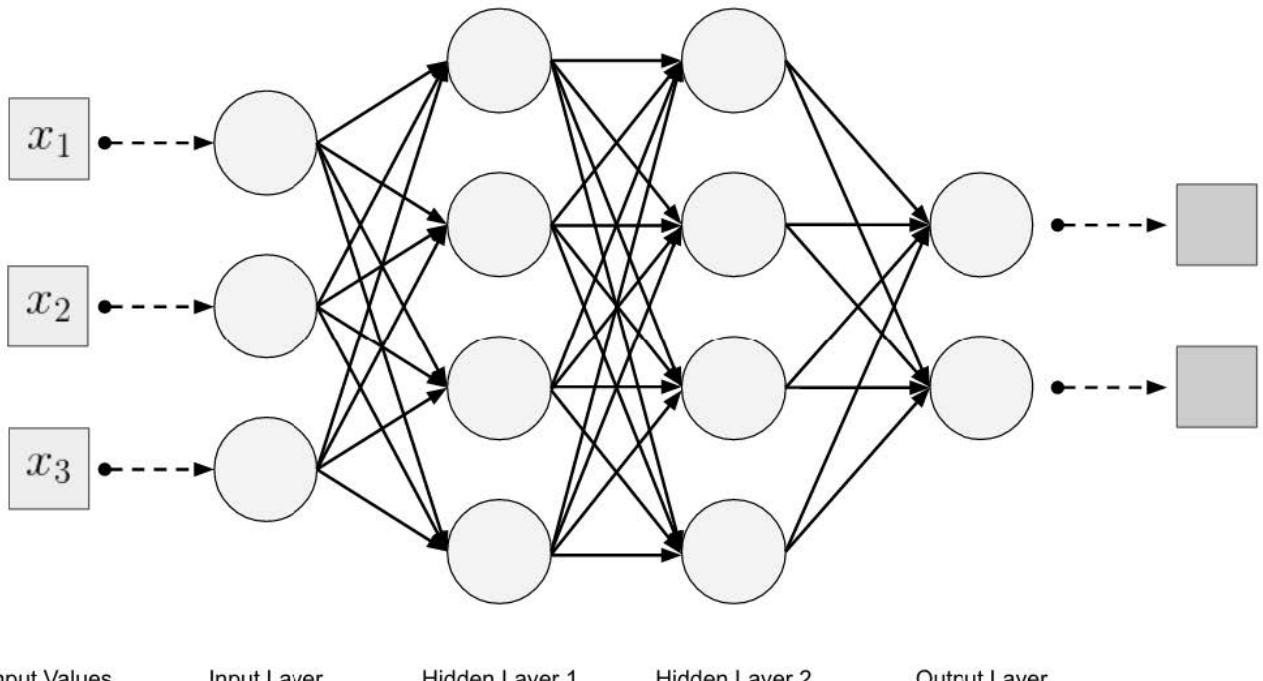
Στα τέλη της δεκαετίας του 50, ο Frank Rosenblatt και πολλοί άλλοι ερευνητές ανέπτυξαν το perceptron. Σε αντιδιαστολή με τον τεχνητό νευρώνα όπου οι παράμετροι του έπρεπε να σχεδιαστούν καθώς δεν υπήρχε κάποια μέθοδος εκπαίδευσης, ο Rosenblatt απέδειξε ότι ο κανόνας εκπαίδευσης (learning rule) που πρότεινε θα συγκλίνει στα σωστά βάρη που λύνουν το προβλημα, εάν αυτά υπάρχουν. Το 1969 ωστόσο οι Marvin Minski και Seymour Papert είπαν ότι ένα Perceptron είναι αδύνατο να "μάθει" να λύνει το πρόβλημα του XOR. Ειδικότερα απέδειξαν θεωρητικά ότι είναι αδύνατο να μάθει να υλοποιεί μία τέτοια συνάρτηση όσο και αν το αφήσουμε να εκπαιδευτεί, πράγμα το οποίο είναι λογικό καθώς το Perceptron μπορεί να υλοποιήσει μόνο γραμμικές συναρτήσεις και η XOR είναι ΜΗ-γραμμική. Η λύση σε αυτό το πρόβλημα ήρθε μετά από αρκετά χρόνια.

Multi Layer Perceptron	2 decision Boundaries
<p>Input Layer Hidden Layer Output Layer</p> 	

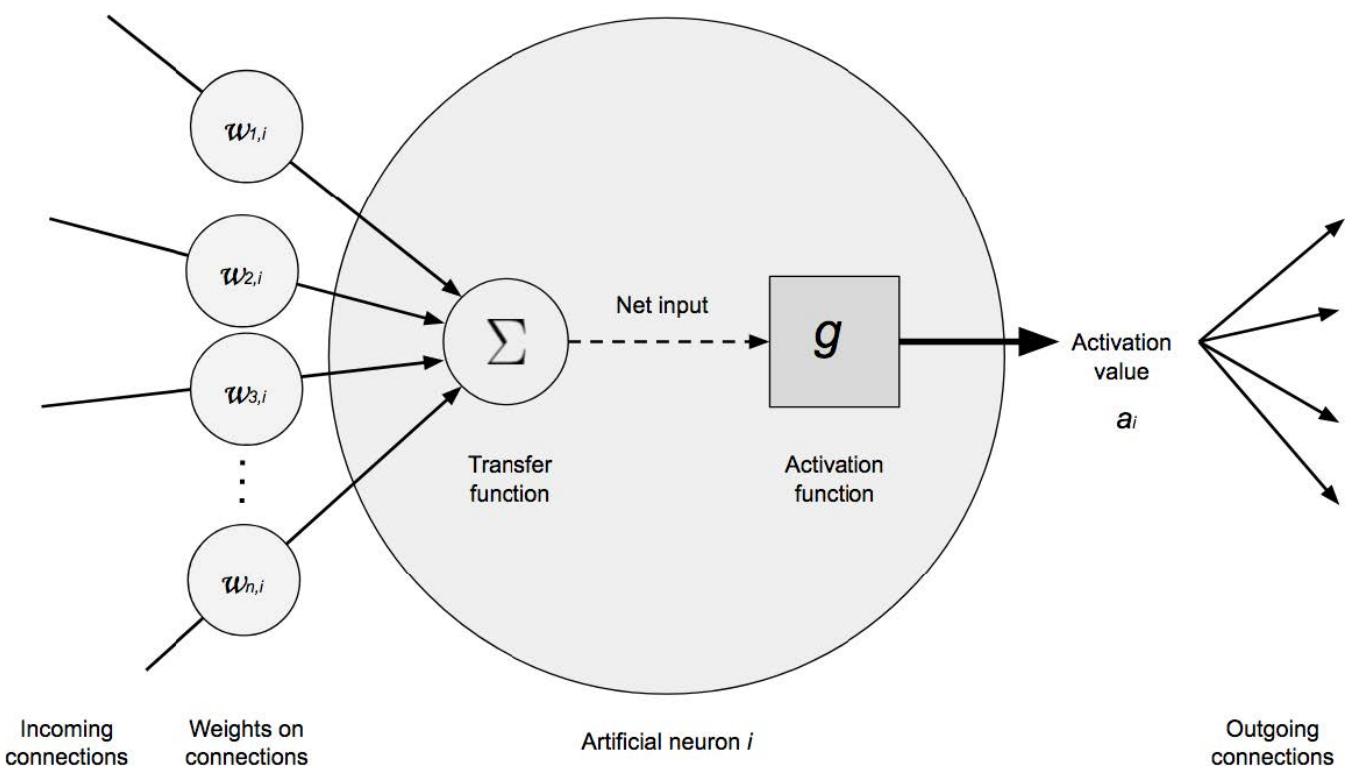
Με βάση το universal approximation theorem μπορούμε να πούμε πως ένα δίκτυο από perceptron μπορεί να χρησιμοποιηθεί για να υλοποιήσει ένα κύκλωμα-συνάρτηση που αποτελείται από πολλές NAND (κάθε πύλη NAND απαιτεί ένα υπερεπίπεδο-γραμμική συνάρτηση). Καθώς ομως οι πύλες NAND είναι καθολικές (Universal) και μπορούν να υλοποιήσουν οποιαδήποτε συνάρτηση, έτσι κατ' αντιστοιχία ένα δίκτυο πολλών (τουλάχιστον δύο) Perceptron μπορεί να προσεγγίσει και αυτό μία οποιαδήποτε συνάρτηση.

Βαθειά Νευρωνικά Δίκτυα

Ένα πολυεπίπεδο νευρωνικό δίκτυο μπορεί να υλοποιηθεί συνενώνοντας πολλούς νευρώνες έτσι ώστε η έξοδος κάποιου νευρώνα να αποτελεί είσοδο σε κάποιον άλλον. Συνήθως ως



Μια πολύ σημαντική προσθήκη σε αυτά που αναφέρθηκαν προηγουμένως είναι η χρήση των Activation functions. Ειδικότερα το σταθμισμένο άθροισμα των εισόδων συν κάποια σταθερά μεροληψίας(bias) που είδαμε ότι υπολογίζει ένας νευρώνας εφαρμόζεται σε μία μη γραμμική συνάρτηση και ως έξοδος πλέον του νευρώνα θεωρείται η έξοδος αυτής της συνάρτησης. Οι μη γραμμικές αυτές συναρτήσεις βοηθάνε ώστε η εκπαίδευση να γίνει σωστά.



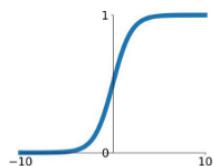
Συναρτήσεις Ενεργοποίησης (Activation Functions)

Υπάρχει μία πληθώρα συναρτήσεων ενεργοποίησης μεταξύ των οποίων ο πιο διαδεδομένες είναι οι εξής:

Activation Functions

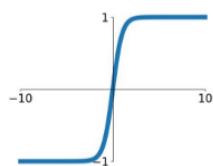
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



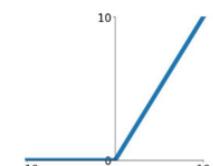
tanh

$$\tanh(x)$$



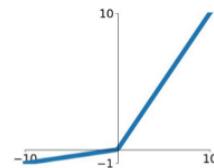
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

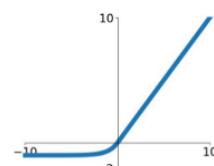


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

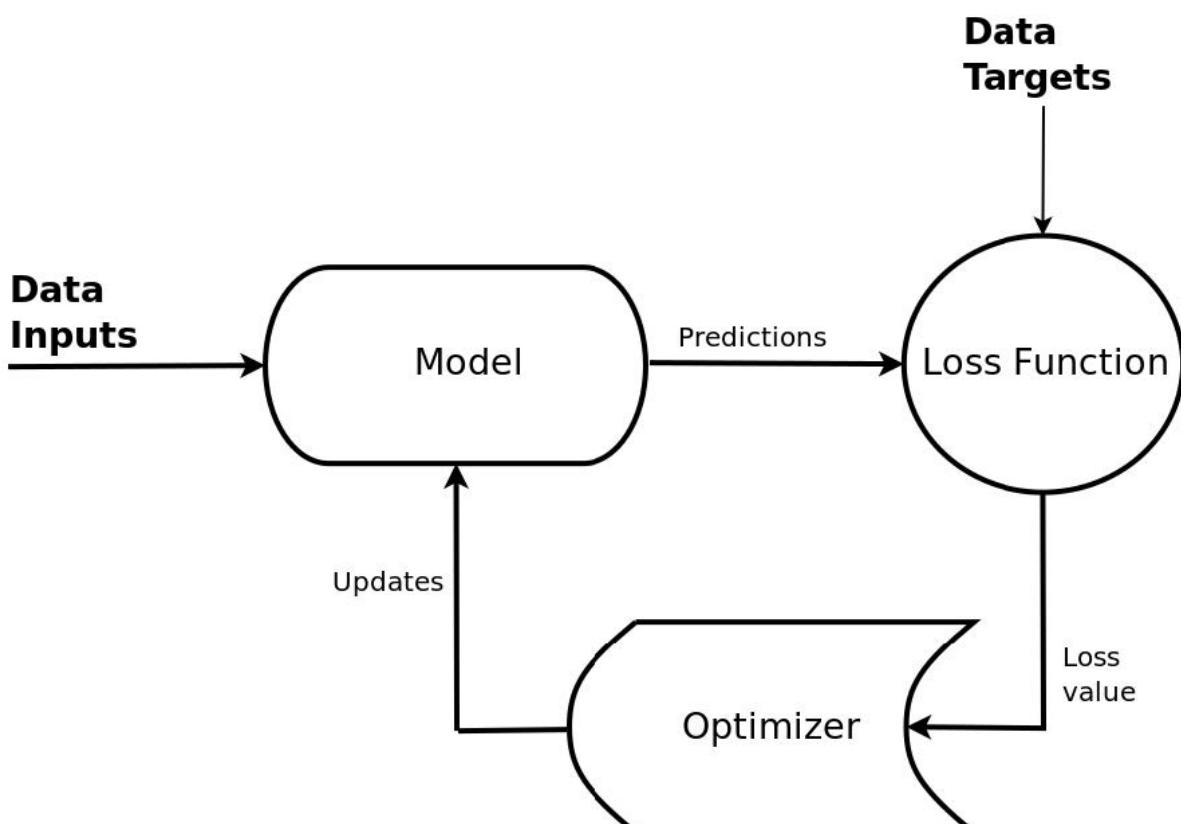
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Μεταξύ αυτών η ευρύτερα χρησιμοποιούμενη είναι η ReLU(Rectified Linear Unit) καθώς και άλλες παραλλαγές της που καλούνται να λύσουν προβλήματα όπως του dying relu.

Επιβλεπόμενη μάθηση(Supervised Learning)

Με τον όρο μάθηση στα Νευρωνικά δίκτυα αναφερόμαστε στην αλλαγή των βαρών των συνδέσεων μεταξύ των νευρώνων με στόχο την καλύτερη ακρίβεια του δικτύου. Ένας τρόπος μάθησης είναι η επιβλεπόμενη μάθηση. Η διαδικασία που ακολουθείται στη συγκεκριμένη περίπτωση συνοψίζεται με το παρακάτω σχήμα.



Μια ακόμη σημαντική ιδέα λοιπόν, είναι αυτή του backpropagation. Μέσω συναρτήσεων κόστους(Cost Function) μπορεί να υπολογιστεί το loss για ένα σύνολο παραδειγμάτων στο forward propagation, και στη συνέχεια χρησιμοποιώντας κάποια μέθοδο βελτιστοποίησης μπορεί να υπολογιστεί πόσο πρέπει να αλλάξουν τα βάρη του δίκτυου και να γίνει αυτή η ανανέωση σε μία διάδοση προς τα πίσω(backpropagation). Ειδικότερα, ένα μοντέλο πταίρνει ένα σύνολο εισόδων, κάνει κάποιες προβλέψεις, τις συγκρίνει με τις πραγματικές τιμές των εξόδων, υπολογίζεται ποσοτικά το σφάλμα μέσω κάποιου loss function και μετά με χρήση κάποιου optimizer που έχει ως στόχο την ελαχιστοποίηση του σφάλματος αυτού ανανεώνονται τα βάρη (με βάση κάποιον κανόνα που ορίζει ο optimizer) και με τη χρήση του backpropagation. Η διαδικασία αυτή πραγματοποιείται πολλές φορές ώστε να φτάσουμε σε βέλτιστο αποτέλεσμα. Πριν περάσουμε στην ανάλυση για τους optimizers και τα loss functions είναι σημαντικό να γινει αναφορά στις έννοιες epoch και batch. Ως **epoch(εποχή)** αναφερόμαστε σε έναν κύκλο όπου όλα τα δεδομένα έχουν περάσει μέσα στο δίκτυο μια φορά για forward propagation και μία για backpropagation. Ως **batch(παρτίδα, ομάδα)** αναφερόμαστε σε ένα μέρος των δεδομένων ου εισάγονται και επεξεργάζονται ταυτόχρονα στο δίκτυο. Συνεπώς η ανανέωση των βαρών γίνεται μετά από κάθε batch.

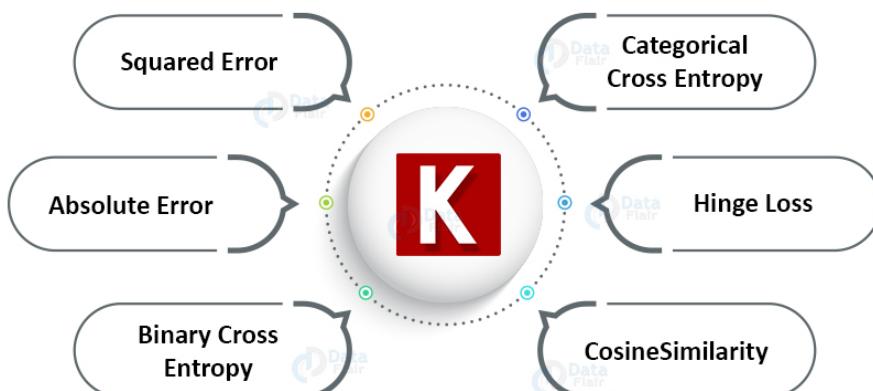
Optimizers - Loss/Cost Functions

Το πιο σημαντικό κομμάτι στην εκπαίδευση είναι η ανανέωση των βαρών. Ο τρόπος με τον οποίο θα ανανεώνονται τα βάρη υπαγορεύεται από τον optimizer. Ο πιο γνωστός και ταυτόχρονα απλός optimizer είναι η Gradient Descend. Συγκεκριμένα η ανανέωση των βαρών γίνεται με βάση των εξής τύπο: $\theta = \theta - \alpha \cdot \nabla J(\theta)$ όπου $J(\theta)$ το σφάλμα που υπολογίζεται μέσω του cost function. Ωστόσο, η ανάγκη για αντιμετώπιση προβλημάτων και μεγαλύτερη ταχύτητα σύγκλισης οδήγησαν στον να σχεδιαστούν νέοι optimizers. Μερικοί γνωστοί optimizers βασισμένοι στον Gradient Descent είναι:

- Stochastic Gradient Descent
- Mini-Batch Gradient Descent(Improvement on both Stochastic Gradient Descent and standard Gradient Descent)
- Momentum -> Update Rule: $\theta = \theta - V(t)$ όπου $V(t) = \gamma V(t-1) + \alpha \cdot \nabla J(\theta)$
- Adagrad -> This optimizer changes the learning rate
- AdaDelta ->It is an extension of **AdaGrad** which tends to remove the *decaying learning Rate* problem of it.
- Adam(Adaptive Moment Estimation)

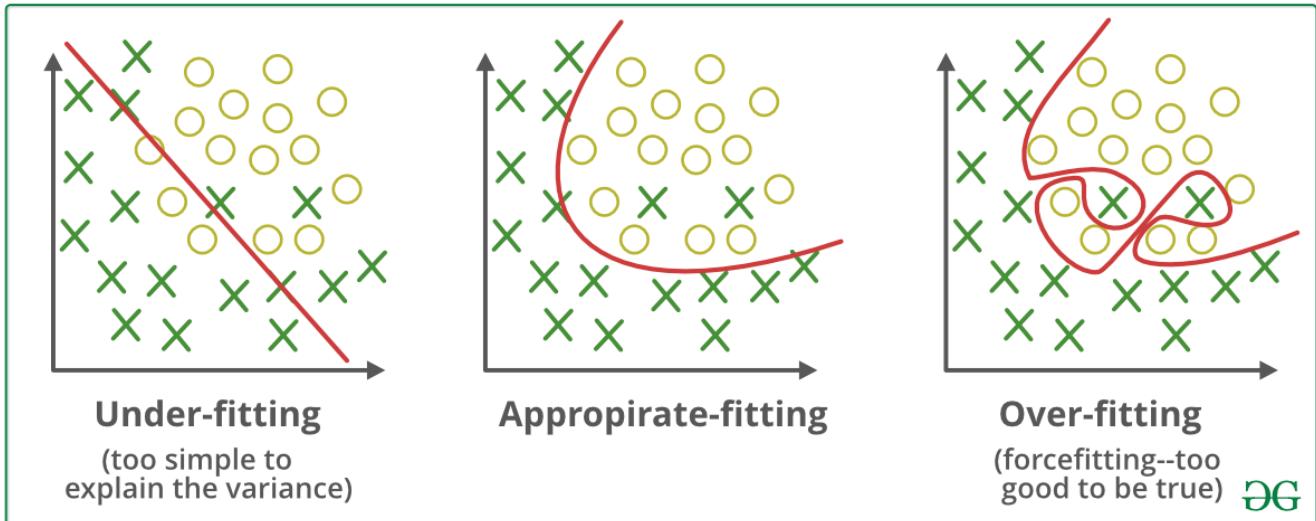
Αξίζει ιδιαίτερα να τονιστεί ότι οι optimizers βασίζονται στην παράγωγο του cost function (το οποίο επηρεάζεται σημαντικά και από το είδος της activation function) καθώς και σε μία παράμετρο η οποία λέγεται Learning rate και συμβολίζεται με το α στον παραπάνω τύπο. Μέσω του learning rate ουσιαστικά ορίζουμε το πόσο μεγάλα βήματα θα κάνουμε και συνεπώς πόσο πολύ θα αλλάξουν τα βάρη. Ειδικότερα πολύ μικρή learning rate μπορεί να οδηγήσει σε τοπικά ελάχιστα που δεν είναι ικανοποιητικά ενώ πολύ μεγάλα βήματα μπορεί να αποτρέψουν και τη συγκλιση. Συνεπώς, η επιλογή του απαιτεί προσοχή. Οσον αφορά τα Cost Functions η επιλογή τους βασίζεται κυρίως στον τύπο δεδομένων και προβλήματος προς επίλυση.

Common Loss & Functions

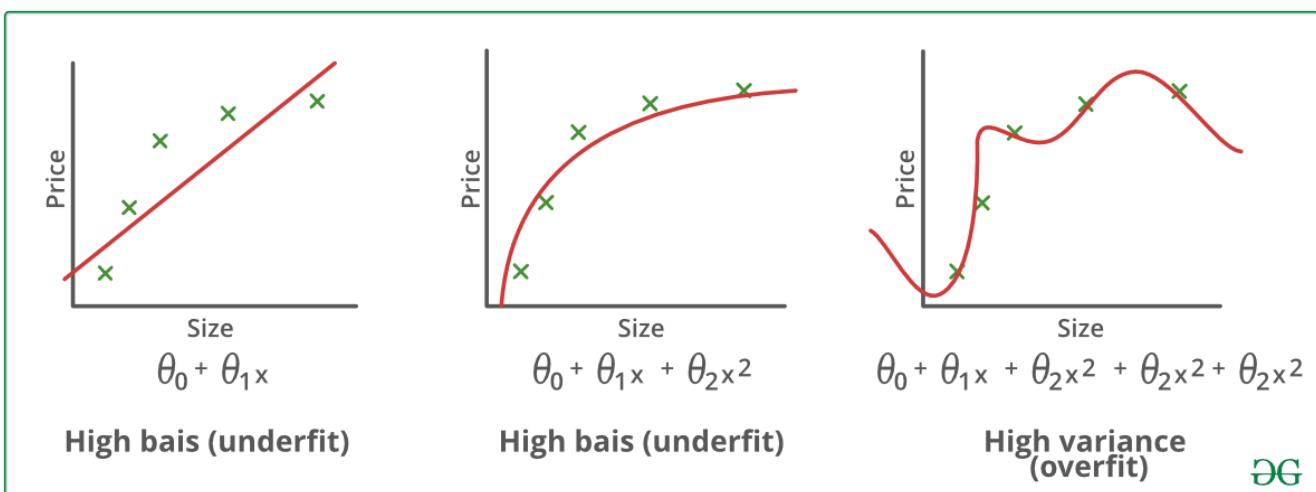


Overfitting - Underfitting

Δύο πολύ σημαντικοί όροι που σχετίζονται με το αποτέλεσμα της εκπαίδευσης είναι η κατάσταση overfit(υπερπροσαρμογής) και underfit(υποπροσαρμογή). Με τον όρο **overfit** εννούμε ότι το μοντέλο αρχίζει και μαθαίνει πολύ καλά τα δεδομένα εκπαίδευσης αλλά χάνει την δυνατότητα να γενικεύει με αποτέλεσμα η απόδοση σε καινούργια δεδομένα που δεν έχει "δει" το δίκτυο να είναι πολύ χαμηλή. Αντίθετα με τον όρο **underfit** εννούμε ότι το μοντέλο δεν έχει έχει μάθει σημαντικά χαρακτηριστικά των δεδομένων και έχει παραμείνει αρκετά απλό.



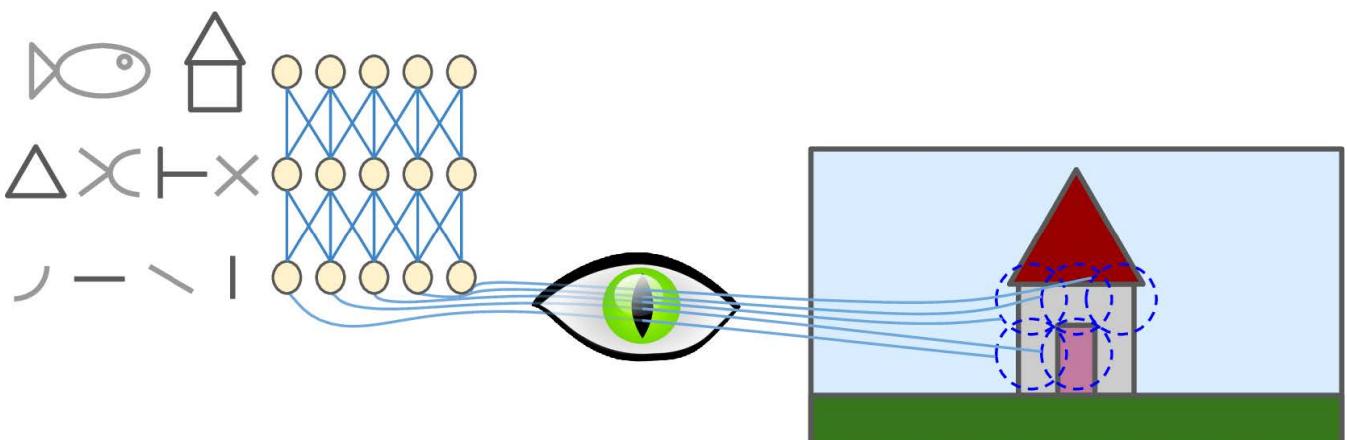
Στα παραπάνω σχήματα φαίνονται ξεκάθαρα οι 3 περιπτώσεις δηλαδή η περίπτωση του overfit, του underfit και η περίπτωση που το μοντέλο έχει κάνει την κατάλληλη προσαρμογή. Στις 2 ακραίες περιπτώσεις είναι αρκετά εύκολο να βρεθεί ένα μοντέλο. Σε κατάσταση overfit από την μία μπορεί να καταλήξουμε σε περίπτωση που ένα πολύ μεγάλο και σύνθετο μοντέλο εκπαιδευται σε ένα πολύ dataset, εάν δεν εφαρμόζεται κάποια μέθοδος regularization και πιθανώς εαν αφήσουμε την εκπαίδευση να γίνει για πάρα πολλές εποχές. Από την άλλη σε κατάσταση underfit μπορεί να βρεθούμε εαν το dataset είναι αρκετά μεγάλο και συνθετο και το μοντέλο είναι σχετικά απλό ή εάν δεν εκπαιδεύτηκε αρκετά. Στην βιβλιογραφία ακόμη εκτός των όρων underfit και overfit θα συναντήσουμε και τους όρους bias και variance. Αυτό συμβαίνει καθώς στην κατάσταση του underfit παρατηρείται αυξημένη τιμή του bias δηλαδή σχετικά μεγάλο training error και αντιστοιχης τάξης validation error ενώ σε κατάσταση overfit παρατηρείται μεγάλη τιμή στη variance (απόκλιση) δηλαδή ένα σχετικά μικρό training error αλλά ενα σημαντικά μεγαλύτερο(πολλαπλάσιο) validation error. Στην περίπτωση του μεγάλου bias δηλαδή έχουμε ένα μοντέλο που αδυνατεί να μάθει βασικά χαρακτηριστικά των δεδομένων ενώ στην περίπτωση της μεγάλης variance το μοντέλο προσαρμόζεται ακόμη και σε μή χρησιμα χαρακτηριστικά η ακόμη και το θόρυβο. Αντίστοιχα, τα διαγράμματα με το bias / variance trade-off φαίνονται παρακάτω.



Συνελικτικά Νευρωνικά Δίκτυα (Convolutional Neural Networks)

Όπως είδαμε προηγουμένως ένας νευρώνας συνδέεται με όλους τους νευρώνες του επόμενου επιπέδου. Αυτό σημαίνει ότι μία τέτοια αρχιτεκτόνική θα ήταν απαγορευτική για μεγάλες εικόνες καθώς θα είχαμε υπορβολικά μεγάλο αριθμό παραμέτρων. Ακόμη, για να τροφοδοτήσουμε σε ένα κλασικό νευρωνικό δίκτυο μια εικόνα πρέπει πρώτα να την μετατρέψουμε σε ένα διάνυσμα. Έτσι, καθώς το νευρωνικό δεν έχει κάποια γνώση για το πώς οργανώνονται τα pixel, χάνεται η χωρική πληροφορία.

Μελέτες στον οπτικό φλοιό(visual cortex) του ανθρώπου έδειξαν ότι πολλοί νευρώνες έχουν ένα τοπικό πεδίο αντίληψης, δηλαδή ανταποκρίνονται σε μια συγκεκριμένη περιοχή του οπτικού πεδίου. Ακόμη, αυτά τα τοπικά πεδία αντίληψης μπορούν να αλληλοεπικαλύπτονται και συνολικά καλύπτουν ολόκληρο το οπτικό πεδίο. Ακόμη, κάποιοι νευρώνες έχουν μεγαλύτερα πεδία αντίληψης και αντιδρούν σε πιο σύνθετα σχήματα/μοτίβα/πρότυπα (patterns). Έτσι, προήλθε η ιδέα ότι η ιδέα ότι σε μία εικόνα αντιλαμβανόμαστε πρώτα low-level χαρακτηριστικά όπως ακμές, γραμμές και απλά σχήματα τα οποία δίνονται ως σε υψηλότερου επιπέδου νευρώνες οι οποίοι μαθαίνουν να αναγνωρίζουν πιο σύνθετα γνωρίσματα. Πολύ σημαντικά ορόσημα στην ιστορία των συνελικτικών δίκτυων είναι το Neocognitron το 1980 και το μοντέλο LeNet που περιγράφεται στο paper των Yann LeCun, Leon Bottou, Yoshua Bengio το 1998 και το AlexNet το 2012.



Στην παραπάνω εικόνα βλεπουμε αυτό που περιγράφηκε προς τα πάνω, δηλαδή ότι από κάτω προς τα πάνω αντιλαμβανόμαστε όλο και πιο σύνθετα γνωρίσματα. Παίρνοντας υπόψη την παραπάνω, τα συνελικτικά νευρωνικά βασίζονται σε 3 βασικές ιδέες:

1. **Local receptive fields**(τοπικά πεδία αντίληψης)
2. **Shared weights** (κοινά βάρη)
3. **Pooling** (υποδειγματοληψία).

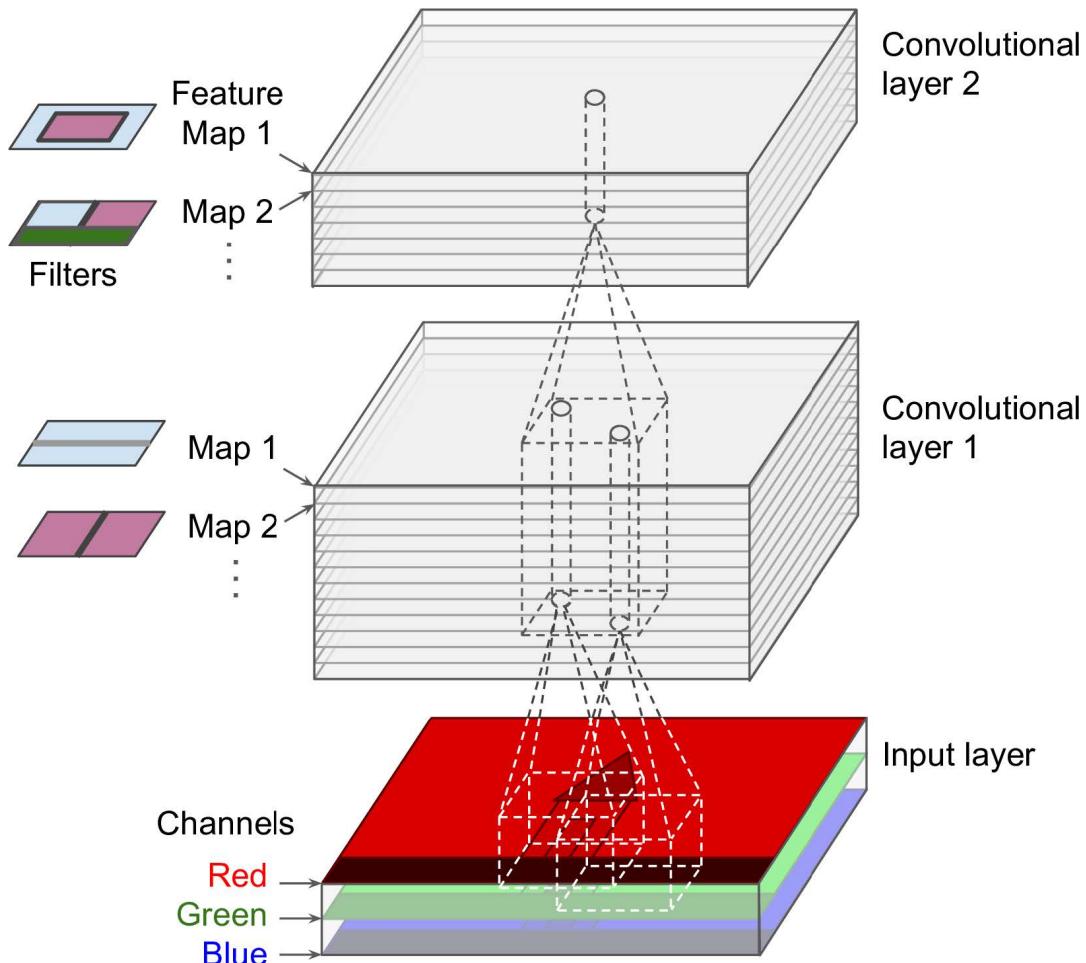
Ως τοπικό πεδίο αντίληψης ενός νευρώνα αναφερόμαστε σε αυτό το παραλληλόγραμμο-τετράγωνο στο παραπάνω σχήμα. Ειδικότερα, σε μία εικόνα μέσω της πράξης της συνέλιξης εφαρμόζονται φίλτρα. Για κάθε τοπικό πεδίο αντίληψης ένας νευρώνας μαθαίνει ένα βάρος και ένα bias. Καθώς το φίλτρο μετακινείται πάνω σε μία εικόνα κατα ένα βήμα ίσο με το stride length. Μέσω των κοινών βαρών και του bias ορίζεται το φίλτρο το οποίο διατρέχει όλη την εικόνα και παράγει αυτό που ονομάζουμε feature map. Εκτός από τη συνέλιξη χρησιμοποιείται και pooling δηλαδή υποδειγματοληψία. Το pooling χρησιμοποιείται μετά τα επίπεδα συνέλιξης και έχει ώστε να συγκεντρώσει/συμπυκνώσει πληροφορία πριν περάσει στο επόμενο επίπεδο συνέλιξης. Συνηθισμένοι τρόποι για pooling είναι το max pooling και το average pooling. Παρακάτω παρουσιάζονται σε σχήματα όσα ήδαμε παραπάνω. Όσον αφορά την πράξη της συνέλιξης ισχυεί ο παρακάτω τύπος.

Equation 14-1. Computing the output of a neuron in a convolutional layer

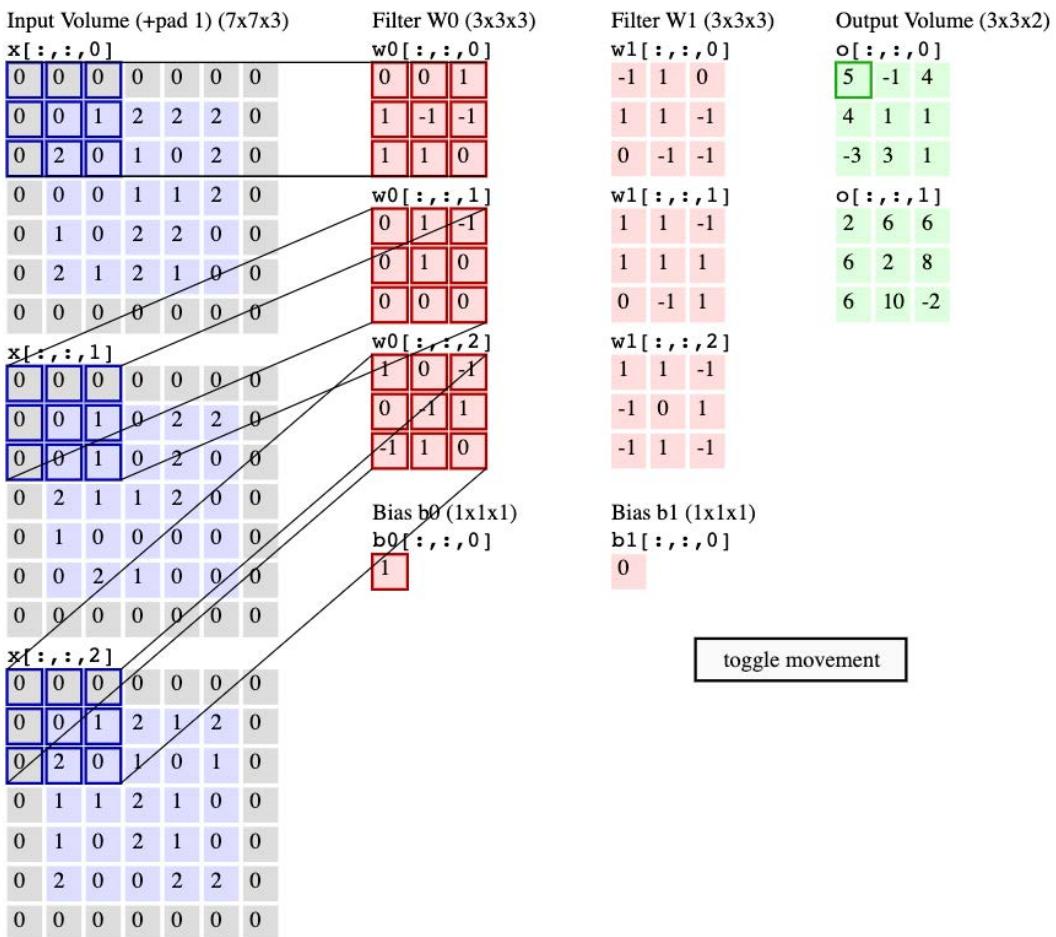
$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{with } \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases}$$

- $z_{i,j,k}$ is the output of the neuron located in row i , column j in feature map k of the convolutional layer (layer l).
- As explained earlier, s_h and s_w are the vertical and horizontal strides, f_h and f_w are the height and width of the receptive field, and $f_{n'}$ is the number of feature maps in the previous layer (layer $l - 1$).
- $x_{i',j',k'}$ is the output of the neuron located in layer $l - 1$, row i' , column j' , feature map k' (or channel k' if the previous layer is the input layer).
- b_k is the bias term for feature map k (in layer l). You can think of it as a knob that tweaks the overall brightness of the feature map k .
- $w_{u,v,k',k}$ is the connection weight between any neuron in feature map k of the layer l and its input located at row u , column v (relative to the neuron's receptive field), and feature map k' .

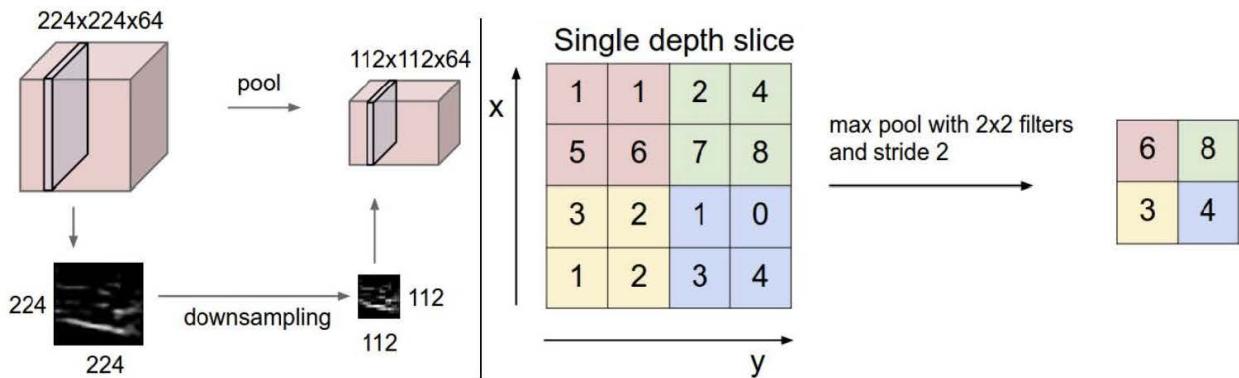
Εδώ παρατηρούμε τα επίπεδα συνέλιξης τα φίλτρα και τα feature maps που παράγονται.



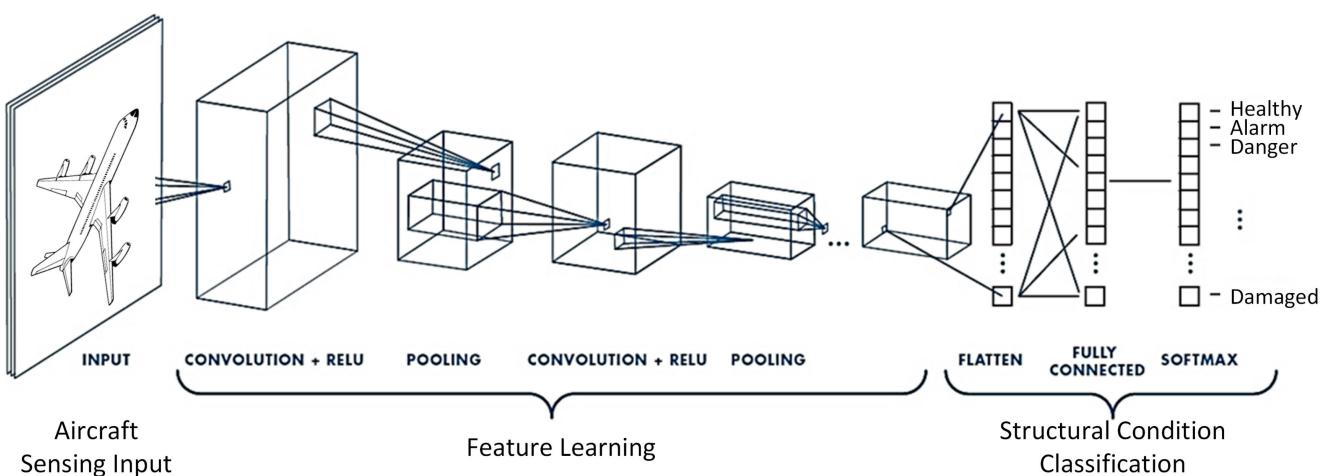
Η συνέλιξη σχηματικά φαίνεται ως εξής:



Αντίστοιχα η πράξη του max-pooling:



Τέλος, αξίζει να αναφέρουμε ότι μια σειρά convolution και pooling απαιτείται ένα τελικό κομμάτι με dense νευρώνες το οποίο λειτουργεί ως classifier.



Fashion Mnist

Άσκηση Multilayer Perceptron

Πρώτο ζητούμενο είναι η δημιουργία ενός Multilayer Perceptron (Fully Connected NN) με 3 κρυφά επίπεδα, 64, 128 και 256, και αρχικό και τελικό επίπεδο κατάλληλου μεγέθους. Οι υπολογισμοί αυτοί γίνονται και στο notebook "Exercise Fashion Mnist.ipynb". Παρακάτω παραθέτω screenshot. Αρχικά, αφού κάνουμε import των κατάλληλων βιβλιοθηκών που θα χρειαστούμε, τυπώνουμε τα μεγέθη του κάθε set (των tensors) κάνουμε οπτικοποίηση ορισμένων εικόνων ώστε να ξέρουμε περίπου τη μορφή τους.

Όσον αφορά το αρχικό, το τελικό και τα ενδιάμεσα επίπεδα έχουμε:

```
Model 1 (Multi Layer Perceptron)

Input layer: 28 · 28 = 784
Hidden Layers: 64, 128, 256
Final layer: 10 (Number of Classes)
```

Στη συνέχεια ορίζουμε το μοντέλο, τον optimizer και ότι άλλο απαιτείται για την εκπαίδευση του(εποχές, batch size, loss function, learning rate). Έτσι παίρνουμε το model summary.

```
[ ] def define_fc_model():
    model = tf.keras.models.Sequential()
    model.add(layers.Flatten(input_shape = (28,28)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dense(256, activation='relu'))
    model.add(layers.Dense(10,activation="softmax"))
    return model

▶ fc_model = define_fc_model()
fc_model.summary()

Model: "sequential_2"
+-----+
Layer (type)          Output Shape         Param #
+-----+
flatten_2 (Flatten)   (None, 784)           0
dense_5 (Dense)       (None, 64)            50240
dense_6 (Dense)       (None, 128)           8320
dense_7 (Dense)       (None, 256)           33024
dense_8 (Dense)       (None, 10)             2570
+-----+
Total params: 94,154
Trainable params: 94,154
Non-trainable params: 0

[ ] fc_model.compile(loss="categorical_crossentropy",
                      optimizer=tf.keras.optimizers.Adam(learning_rate=5e-4),
                      metrics=['acc'])

[ ] fc_model.fit(x_train, y_train,
                  batch_size=64,
                  epochs=25,
                  verbose=1)
```

Τα αποτελέσματα είναι τα εξής:

```
Epoch 25/25
938/938 [=====] - 4s 4ms/step - loss: 0.1546 - acc: 0.9406
<keras.callbacks.History at 0x7f312761fa90>

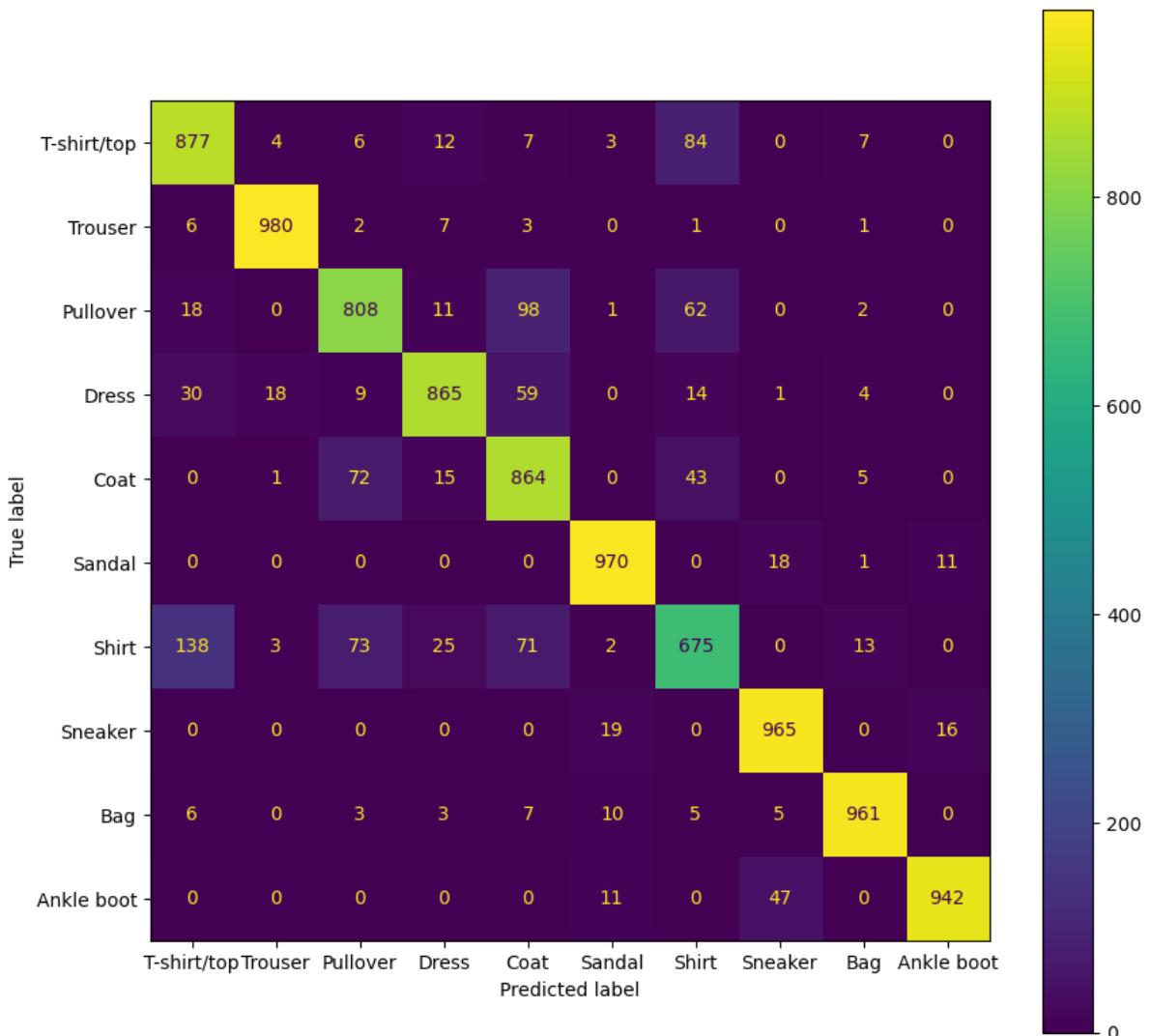
predictions_fc=fc_model.predict(x_test)
predicted_classes_fc=np.argmax(predictions_fc, axis=1)
np.save("Predicted_FM_FC.npy", predicted_classes_fc)
y_test_values = np.argmax(y_test, axis=-1) # There are in categorical form(One hot encoded)
np.save("Y_test_values.npy", y_test_values)
print(classification_report(y_test_values, predicted_classes_fc, target_names=class_names,digits=5))

      precision    recall  f1-score   support

T-shirt/top      0.81581   0.87700   0.84530     1000
  Trouser       0.97416   0.98000   0.97707     1000
 Pullover       0.83842   0.80800   0.81906     1000
    Dress        0.92217   0.86500   0.89267     1000
    Coat         0.77908   0.86400   0.81935     1000
  Sandal        0.95472   0.97000   0.96230     1000
   Shirt        0.76357   0.67500   0.71656     1000
Sneaker        0.93147   0.96500   0.94794     1000
    Bag          0.96680   0.96100   0.96389     1000
Ankle boot      0.97214   0.94200   0.95683     1000

  accuracy      0.89070   0.89070   0.89070     10000
macro avg      0.89103   0.89070   0.89010     10000
weighted avg   0.89103   0.89070   0.89010     10000
```

Ειδικότερα, όπως βλέπουμε έχουμε 94.06% στο training set και 89.07% στο test set. Αντίστοιχα το confusion matrix είναι:



Άσκηση CNN

Σε αυτή την άσκηση, ζητείται να δημιουργήσουμε ένα CNN με 2 κρυφά επίπεδα με max pooling μετά από κάθε ένα, με 32 και 64 φίλτρα. Αντίστοιχα με πρίν λοιπόν έχουμε το μοντέλο και το model summary. Ο κώδικας βρίσκεται και πάλι στο "Exercise Fashion Mnist.ipynb".

Model 2 (Convolutional Neural Network)

Hidden Layers 32,64 filters

```
[ ] def define_cnn_model():
    model = tf.keras.models.Sequential()
    model.add(layers.Conv2D(32, (5, 5), activation="relu", padding="same", input_shape=(28,28,1)))
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2,2)))
    model.add(layers.Conv2D(64, (3, 3), activation="relu", padding="same"))
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2,2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(10,activation="softmax"))
    return model
```

```
▶ cnn_model = define_cnn_model()
cnn_model.summary()
```

```
□ Model: "sequential_4"
```

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_4 (Conv2D)	(None, 28, 28, 32)	832
<hr/>		
max_pooling2d_4 (MaxPooling 2D)	(None, 14, 14, 32)	0
conv2d_5 (Conv2D)	(None, 14, 14, 64)	18496
max_pooling2d_5 (MaxPooling 2D)	(None, 7, 7, 64)	0
flatten_4 (Flatten)	(None, 3136)	0
dense_10 (Dense)	(None, 10)	31370
<hr/>		
Total params:	50,698	
Trainable params:	50,698	
Non-trainable params:	0	

Τα αποτελέσματα είναι τα εξής:

```
Epoch 25/25
938/938 [=====] - 7s 7ms/step - loss: 0.1161 - acc: 0.9579
<keras.callbacks.History at 0x7f30ab9b8fd0>
```

```
▶ predictions_cnn=cnn_model.predict(x_test)
predicted_classes_cnn=np.argmax(predictions_cnn,axis=1)
y_test_values = np.argmax(y_test, axis=-1) # There are in categorical form(One hot encoded)
print(classification_report(y_test_values, predicted_classes_cnn, target_names=class_names,digits=5))
```

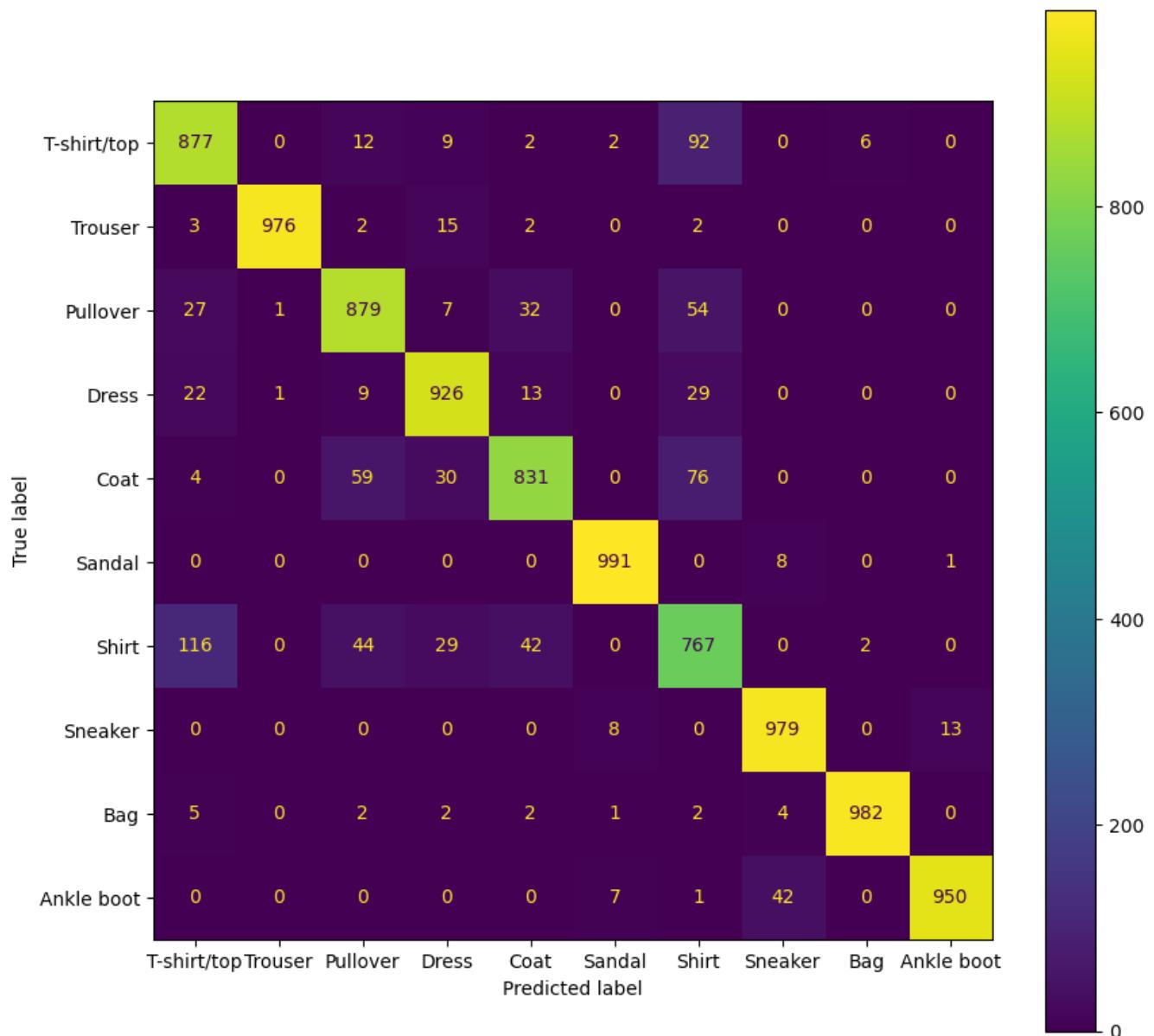
```
□          precision    recall  f1-score   support

T-shirt/top      0.83207   0.87700   0.85394     1000
Trouser        0.99796   0.97600   0.98686     1000
Pullover       0.87289   0.87900   0.87593     1000
Dress          0.90963   0.92600   0.91774     1000
Coat            0.89935   0.83100   0.86383     1000
Sandal         0.98216   0.99100   0.98656     1000
Shirt           0.74976   0.76700   0.75828     1000
Sneaker         0.94773   0.97900   0.96311     1000
Bag              0.99192   0.98200   0.98693     1000
Ankle boot     0.98548   0.95000   0.96741     1000

accuracy           0.91580      0.91580     0.91580     10000
macro avg        0.91689   0.91580   0.91606     10000
weighted avg     0.91689   0.91580   0.91606     10000
```

Στη περίπτωση αυτή όπως ήταν αναμενόμενο για ίδιο αριθμό εποχών με πρίν έχουμε καλύτερα αποτελέσματα.

Ειδικότερα στο training set έχουμε 95.79% ενώ στο test set 91.58%. Αξίζει να σημειωθεί ότι τα αποτελέσματα αυτά είναι καλύτερα ενώ το δίκτυο έχει σχεδόν τις περίπου τις μισές παραμέτρους σε σχέση με το fully connected. Τέλος, παρουσιάζεται και πάλι το confusion matrix.

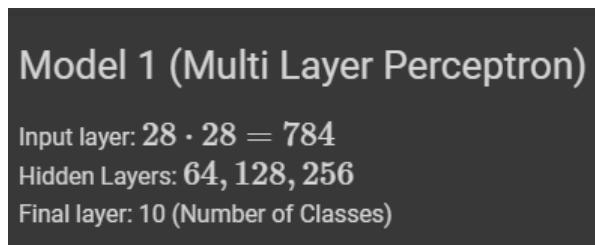


Cifar - 10

Άσκηση Multilayer Perceptron

Αντίστοιχα με την προηγούμενη άσκηση ζητείται ένα Multilayer Perceptron (Fully Connected NN) με 3 κρυφά επίπεδα, 64, 128 και 256, και αρχικό και τελικό επίπεδο κατάλληλου μεγέθους. Το αρχείο του notebook είναι το **Exercise Cifar-10.ipynb**.

Όσον αφορά το αρχικό, το τελικό και τα ενδιάμεσα επίπεδα έχουμε:



Στη συνέχεια ορίζουμε το μοντέλο και ότι άλλο απαιτείται για την εκπαίδευση και παρουσιάζεται το model summary.

Τα αποτελέσματα είναι τα εξής:

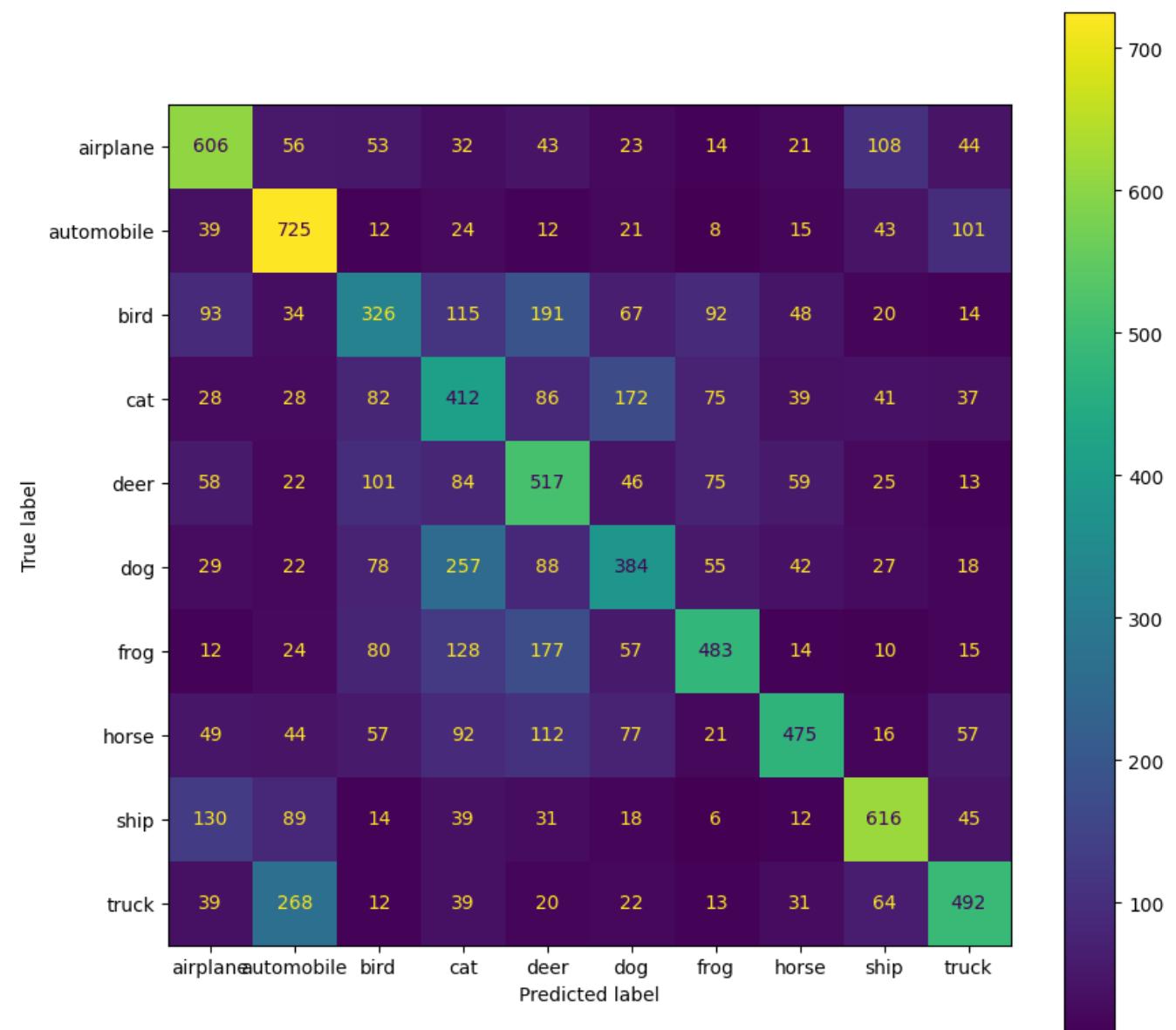
```
[ Epoch 20/20
782/782 [=====] - 3s 4ms/step - loss: 1.2692 - acc: 0.5467
<keras.callbacks.History at 0x7f8ba4693d50>

▶ predictions_fc=fc_model.predict(x_test)
predicted_classes_fc=np.argmax(predictions_fc, axis=1)
y_test_values = np.argmax(y_test, axis=-1) # There are in categorical form(One hot encoded)
print(classification_report(y_test_values, predicted_classes_fc, target_names=class_names,digits=5))

precision    recall   f1-score   support
airplane     0.54590  0.55900  0.55237    1000
automobile   0.62664  0.62100  0.62381    1000
bird         0.36120  0.43200  0.39344    1000
cat          0.33479  0.38200  0.35684    1000
deer         0.49425  0.38700  0.43410    1000
dog          0.42728  0.37900  0.40170    1000
frog         0.53045  0.54000  0.53518    1000
horse        0.74031  0.38200  0.50396    1000
ship         0.58520  0.68000  0.62905    1000
truck        0.50156  0.64300  0.56354    1000

accuracy      0.50050
macro avg    0.51476  0.50050  0.49940    10000
weighted avg  0.51476  0.50050  0.49940    10000
```

Ειδικότερα, όπως βλέπουμε έχουμε 54.5% στο training set και 50.36% στο test set. Αντίστοιχα το confusion matrix είναι:



Άσκηση CNN

Σε αυτή την άσκηση, ζητείται να δημιουργήσουμε ένα CNN με 3 κρυφά επίπεδα με max pooling μετά από κάθε ένα, με 32, 64 και 128 φίλτρα. Αντίστοιχα με πρίν λοιπόν έχουμε το μοντέλο και το model summary. Ο κώδικας βρίσκεται και πάλι στο **Exercise Cifar-10.ipynb**.

```
▶ def define_cnn_model():
    model = tf.keras.models.Sequential()
    model.add(layers.Conv2D(32, (5, 5), activation="relu", padding="same", input_shape=(32,32,3)))
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2,2)))
    model.add(layers.Conv2D(64, (3, 3), activation="relu", padding="same"))
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2,2)))
    model.add(layers.Conv2D(128, (3, 3), activation="relu", padding="same"))
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2,2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(10, activation="softmax"))
    return model

[ ] cnn_model = define_cnn_model()
cnn_model.summary()

Model: "sequential_1"
-----  

Layer (type)          Output Shape         Param #
-----  

conv2d (Conv2D)        (None, 32, 32, 32)      2432  

max_pooling2d (MaxPooling2D) (None, 16, 16, 32)      0  

)  

conv2d_1 (Conv2D)       (None, 16, 16, 64)      18496  

max_pooling2d_1 (MaxPooling2D) (None, 8, 8, 64)      0  

2D)  

conv2d_2 (Conv2D)       (None, 8, 8, 128)      73856  

max_pooling2d_2 (MaxPooling2D) (None, 4, 4, 128)      0  

2D)  

flatten_1 (Flatten)     (None, 2048)           0  

dense_4 (Dense)         (None, 10)             20490  

-----  

Total params: 115,274  

Trainable params: 115,274  

Non-trainable params: 0
```

Τα αποτελέσματα είναι:

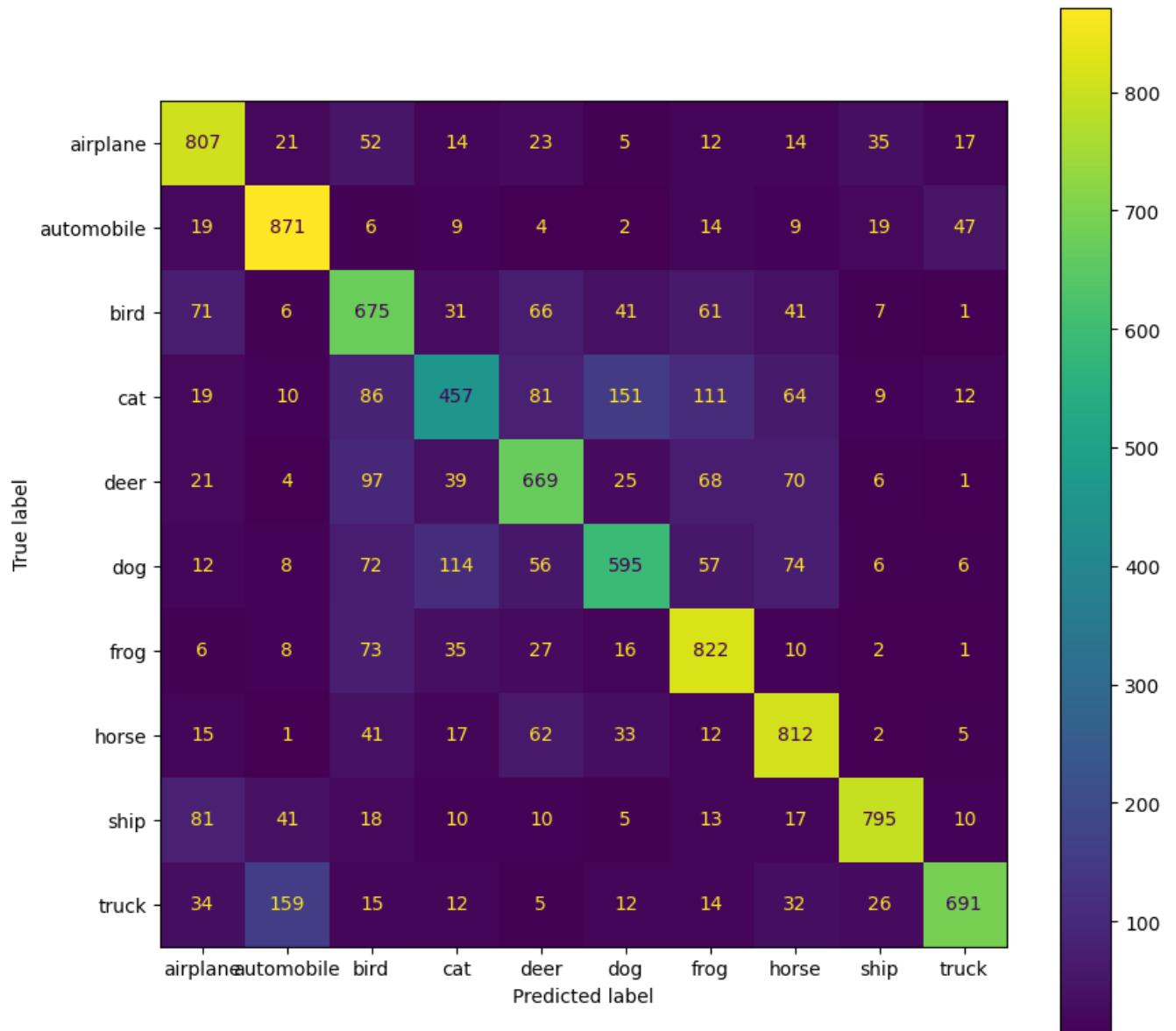
```
Epoch 20/20
782/782 [=====] - 8s 10ms/step - loss: 0.3049 - acc: 0.8950
<keras.callbacks.History at 0x7f8ba43ee810>

▶ predictions_cnn=cnn_model.predict(x_test)
predicted_classes_cnn=np.argmax(predictions_cnn, axis=1)
y_test_values = np.argmax(y_test, axis=-1) # There are in categorical form(One hot encoded)
print(classification_report(y_test_values, predicted_classes_cnn, target_names=class_names,digits=5))

precision    recall   f1-score   support
airplane      0.74378   0.80700   0.77410      1000
automobile    0.77148   0.87100   0.81822      1000
bird          0.59471   0.67500   0.63232      1000
cat           0.61924   0.45700   0.52589      1000
deer          0.66700   0.66900   0.66800      1000
dog           0.67232   0.59500   0.63130      1000
frog          0.69426   0.82200   0.75275      1000
horse         0.71041   0.81200   0.75782      1000
ship          0.87652   0.79500   0.83377      1000
truck         0.87358   0.69100   0.77164      1000

accuracy          0.71940      10000
macro avg       0.72233   0.71940   0.71658      10000
weighted avg    0.72233   0.71940   0.71658      10000
```

Στη περίπτωση αυτή όπως ήταν αναμενόμενο και πάλι για ίδιο αριθμό εποχών με πρίν έχουμε καλύτερα αποτελέσματα. Ειδικότερα στο training set έχουμε 89.5% ενώ στο test set 71.94%. Αξίζει να σημειωθεί ότι τα αποτελέσματα αυτά είναι καλύτερα ενώ το δίκτυο έχει σχεδόν τις περίπου τις μισές παραμέτρους σε σχέση με το fully connected. Σε αυτή τη περίπτωση όπου το dataset είναι πιο σύνθετο από το fashion mnist παρατηρούμε ότι η διαφορά του fully connected με το cnn είναι της τάξης του 20% ακόμη και για απλά δίκτυα. Τέλος εδώ βλέπουμε πως το training accuracy είναι πολύ υψηλότερο το testing πράγμα που φανερώνει ότι πιθανότατα το δίκτυο να άρχισε να κάνει overfit. Στη περίπτωση αυτή δε χρησιμοποιήσαμε κάποια τεχνική(στη δική μου υλοποίηση θα δοκιμαστούν). Το confusion matrix είναι:

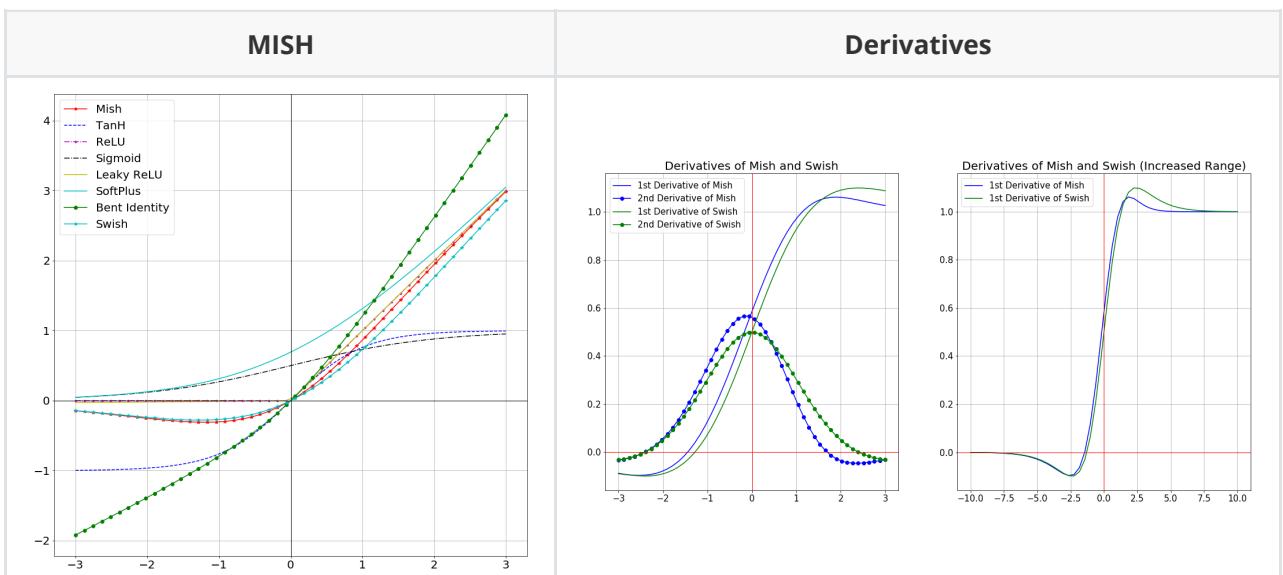


Νευρωνικά Δίκτυα δικής μου έμπνευσης

Πριν παρουσιαστούν τα νευρωνικά δίκτυα δικής μου έμπνευσης για κάθε dataset αξίζει να γίνει μία αναφορά στις επιπλέον μεθόδους και τεχνικές χρησιμοποιήθηκαν.

- Validation Set:** To validation set η αλλιώς τα δεδομένα επικύρωσης, αποτελεί μέρος των δεδομένων το οποίο χρησιμοποιείται για την ρύθμιση των υπερπαραμέτρων του δίκτυου. Ειδικότερα σε κάθε εποχή το μοντέλο το οποίο κάνει fit στο training set αξιολογείται στο validation set. Το validation set δηλαδή επηρεάζει έμμεσα το μοντέλο αλλά όμεσα καθώς η ανανέωση των βαρών γίνεται χρησιμοποιώντας το training set. Στα πλαίσια της εργασία χρησιμοποιείται μέρος του αρχικού training ως validation. Αξίζει να σημειωθεί πως υπάρχει και η μέθοδος του cross validation η οποία είναι πολύ πιο αξιόπιστη ειδικά για να βγουν συμπεράσματα για δοκιμές υπερπαραμέτρων η νέων συναρτήσεων ενεργοποίησης και optimizer αλλά δε χρησιμοποιείται στην συγκεκριμένη εργασία. Τέλος, τυπικές τιμές για το μέγεθος του validation είναι το 10-20% του training set.
- Data Augmentation:** Με τον όρο data augmentation, η αλλιώς επαύξηση δεδομένων αναφερόμαστε στη διαδικασία που έχει στόχο να αυξήσει την ποικιλομορφία του training set εφαρμόζοντας διάφορους μετασχηματισμούς, όπως για παράδειγμα περιστροφή(rotation), μετατόπιση κατακορυφή/οριζοντια, ζουμ κτλπ. Ουσιαστικά κατα τη διάρκεια της εκπαίδευσης εφαρμόζονται μετασχηματισμοί live στα δεδομένα training με κάποια πιθανότητα, με στόχο το μοντέλο να αποκτήσει καλύτερη δυνατότητα γενίκευσης. Αξίζει να σημειωθεί ότι τα δεδομένα αυτά δεν παράγονται και προστίθενται έξτρα στο training set. Για παράδειγμα όταν έρχεται μια εικόνα από το training, εφαρμόζονται μετασχηματισμοί, και το δίκτυο παίρνει στη είσοδο του την εικόνα με τους μετασχηματισμούς. Ακόμη, όπως προαναφέρθηκε σε κάθε εποχή εφαρμόζονται τυχαία αυτοί οι μετασχηματισμοί. Τέλος, πρέπει να πούμε πως αυτή η μέθοδος θέλει προσοχή και μελέτη του συνόλου δεδομένων πριν τη εφαρμογή της(για να δούμε ποίοι μετασχηματισμοί πιθανώς θα βιοθούσαν) καθώς εαν γίνει κατάχρηση τότε τα αποτελέσματα θα είναι χειρότερα αντί για καλύτερα.
- Δοκιμές διαφορετικών συναρτήσεων ενεργοποίησης(swish, mish):** Στα πλαίσια των δοκιμών ορίσα και 2 custom activation functions. Ειδικότερα ορίστηκαν οι συναρτήσεις ενεργοποίησης mish και swish. Μετά από δοκιμές κατέληξα πως η mish (η οποία θα περιγραφεί στην συνέχεια) είχε τα καλύτερα αποτελέσματα και από την swish και την ReLU τις περισσότερες φορές αρκεί να χρησιμοποιηθεί χαμηλότερο learning rate όπως προτείνεται και από τους συγγραφείς του paper. Να σημειωθεί πως τόσο η mish όσο και η swish είναι παραλλαγές της ReLU. Τόσο για την swish όσο και για την mish παρατίθενται τα paper τα οποία της αναλύουν καθώς επίσης παρακάτω παραθέτω και 2 εικόνες από τα loss Landscape που παράγεται και συγκρίνονται ReLU, Swish, Mish. Αξίζει, ωστόσο να σημειωθεί πως μία πιο έμπιστη μέθοδος για να επικυρώσουμε πως όντως αποδίδει καλύτερα η mish είναι αυτή του K - Fold Cross Validation.

Όπως βλέπουμε και στον κώδικα η mish ορίζεται ως εξής: $f(x) = x \cdot \tanh \cdot (\text{softplus}(x))$



Costum implementation for Swish activation function

Official Paper: <https://arxiv.org/pdf/1710.05941v1.pdf>

```
[ ] from keras.backend import sigmoid
def swish(x, beta = 1):
    return (x * sigmoid(beta * x))
from keras.utils.generic_utils import get_custom_objects
get_custom_objects().update({'swish': swish})

# It is already implemented at keras.activations.swish()
```

Custom implementation for Mish activation function

Official Paper: <https://arxiv.org/pdf/1908.08681.pdf>

```
[ ] from keras.backend import tanh,softplus
def mish(x):
    return x * tanh(softplus(x))
from keras.utils.generic_utils import get_custom_objects
from keras.layers import Activation
get_custom_objects().update({'mish':mish})

# It is already implemented at tfa.activations.mish()
```

Εδώ αξίζει να δούμε και τις διαφορές στα loss landscape (Έχουν παραχθεί με το δίκτυο ResNet - 20 για το dataset CIFAR 10 για 200 εποχές από τους συγγραφείς του πείπερ).



4. **Callbacks:** Γενικά με τον όρο callbacks αναφερόμαστε σε ένα αντικέιμενο το οποίο μπορεί να εκτελεί διάφορες ενέργειες κατά τη διάρκεια της εκπαίδευσης, πχ στην αρχή ή στο τέλος κάθε εποχής.

- **Model Checkpoint:** Το modelCheckpoint χρησιμοποιείται σε συνδιασμό με το model.fit() για να αποθηκεύονται τα βάρη ενός μοντέλου (σε ένα αρχείο checkpoint) σε συγκεκριμένες χρονικές στιγμές. Ειδικότερα μέσω κατάλληλων επιλογών, εγώ κρατάω κάθε φορά τα βάρη όταν το δηλαδή το μοντέλο έχει καλύτερη απόδοση ως προς την μετρική του validation loss, δηλαδή όταν αυτό μειώνεται.
- **Early Stopping:** Όπως προαναφέρθηκε ένα από τα προβλήματα που εμφανίζονται συχνά κατά την εκπαίδευση των νευρωνικών δικτύων είναι το overfitting. Ένας τρόπος για να αποφύγουμε το φαινόμενο αυτό είναι να σταματάμε την εκπαίδευση έγκαιρα. Ωστόσο, είναι αρκετά δύσκολο να καθορίσουμε πόσες εποχές ακριβώς χρειάζεται ένα μοντέλο. Μία λύση είναι λοιπόν να χρησιμοποιήσουμε έναν αρχικά μεγάλο αριθμό εποχών σε συνδιασμό με το early stopping, το οποίο θα τερματίσει την εκπαίδευση όταν validation loss (η γενικά η μετρική που επιλέγουμε) παύει να βελτιώνεται.

- **ReduceLROnPlateau:** Η τεχνική αυτή υπάγεται στην κατηγορία του Decaying Learning Rate, δηλαδή στη μείωση του learning rate κατά τη διάρκεια της εκπαίδευσης. Ειδικότερα, ως γνωστόν, το learning rate αποτελεί υπερπαράμετρο στην εκπαίδευση. Πολύ εγάλες τιμές μπορούν να οδηγήσουν σε αστάθειες ενώ πολύ μικρές μπορεί να καθυστερούν δραματικά την εκπαίδευση η ακόμη να οδηγήσουν σε ένα κακό τοπικό ελάχιστο. Μια αρκετά καλή τεχνική λοιπόν είναι να ξεκινάει η εκπαίδευση με πιο μεγάλες τιμές για learning rate έτσι ώστε να αποφευχθούν κακά τοπικά ελάχιστα και στη συνέχεια να αρχίσουμε να μειώνουμε το learning rate ώστε να αποφύγουμε φαινόμενα αστάθιας. Στη συγκεκριμένη τεχνική, δηλαδή του ReduceLROnPlateau, το learning rate μειώνεται (έχοντας προφανώς ενα κάτω όριο) κάθε φορά που η εκπαίδευση τείνει να σταματήσει, δηλαδή δεν έχουμε βελτίωση της μετρικής που ορίσαμε.
- **Tensorboard:** Το tensorboard μας επιτρέπει να κρατάμε logs κατά την εκπαίδευση και μετά να πάρουμε πολλές πληροφορίες και να κάνουμε plots. Ειδικότερα μπορούμε να δούμε μέσω των αντιστοιχων plot, πως μεταβάλονται το loss και το accuracy κατά τη διάρκεια της εκπαίδευσης(metrics summary plots), τα activation histograms και πολλά άλλα.

5. **Regularization Methods:** Όπως αναφέρθηκε ήδη πολλές φορές ένα από τα προβλήματα που εμφανίζονται συχνά κατά την εκπαίδευση των νευρωνικών δικτύων είναι το overfitting. Στα πλαίσια αυτού έχουν αναπτυχθεί και προταθεί πολλές τεχνικές για βελτίωση της ικανότητας γενίκευσης των νευρωνικών δικτύων. Παρακάτω παρουσιάζω κάποιες που χρησιμοποίησα:

- **Batch Normalization:** Ένα από τα προβλήματα που εμφανίζονται κατά την εκπαίδευση των νευρωνικών, είναι η συνεχής αλλαγή της εισόδου των κρυφών επιπέδων και κατ επέκταση των κατανομών αυτών των εισόδων. Αυτή η αλλαγή ονομάζεται διακυμαίνουσα μετατόπιση(interal co-variate shift) και είναι ένας από τους λόγους που κατα τη διάρκεια της εκπαίδευσης των νευρωνικών δικτύων είναι απαραίτητη η μείωση του ρυθμού εκμάθησης, ώστε να συνεχίσει να μειώνεται το κόστος. Ένας προτινόμενος τρόπος για αντιμετώπιση του προβλήματος είναι η κανονικοποίηση παρτίδας (Batch Normalization). Γίνεται κανονικοποίηση δηλαδή βάση των στατιστικών της παρτίδας για κάθε επίπεδο, έτσι ώστε ο μέσος όρος να είναι 0 και η τυπική απόκλιση 1.
- **Dropout:** Η τεχνική της απόσυρσης(Dropout) δημιουργήθηκε έτσι ώστε να αντιμετωπιστεί αποτελεσματικά το πρόβλημα του overfitting. Ειδικότερα κατα τη διάρκεια της εκπαίδευσης ένα τυχαίο πλήθος νευρώνων που ορίζουμε εμείς απενεργοποιούνται διαδίδοντας μηδενικά στην έξοδό τους ενώ οι υπόλοιποι διατηρούνται ενεργοποιημένοι.
- **Layer weight Regularizers:** Όταν ένα δίκτυο έχει μεγάλα βάρη, αυτό μπορεί να αποτελέι ένδειξη ότι το δίκτυο είναι ασταθές καθώς μικρές αλλαγές στις ειδόδους του δικτύου θα μπορούσαν να οδηγήσουν σε τεράστιες αλλαγές στην έξοδο, φαινόμενο που συχνά εμφανίζεται όταν υπάρχει overfitting. Ένας τρόπος για να αποφύγουμε το φαινόμενο αυτό είναι να τιμωρούμε το δίκτυο με βάση το μέγεθος των βαρών, προσθέτοντας έναν έξτρα παράγοντα στο loss function. Έτσι, ωθούμε το δίκτυο να διατηρήσει χαμηλές τιμές βαρών.

Νευρωνικό δικής μου έμπνευσης για Fashion Mnist

Τα αποτελέσματα που κράτησα για το Fashion Mnist στο test set κυμαίνονται από 92.9% έως 94.19%. Παρακάτω θα παρουσιαστεί το notebook και το δίκτυο για το μέγιστο accuracy. Το αρχείο του notebook ονομάζεται **Beautiful_Net.ipynb**. Ωστόσο, στο τέλος θα υπάρχει σύνδεσμος για google drive που μπορούν να βρεθούν και τα υπόλοιπα notebooks σε υποφακέλους (καθώς έχουν όλα το ίδιο όνομα αλλά στο όνομα του φακέλου αναφέρεται και το accuracy).

Η τελική αρχιτεκτονική με στόχο το μέγιστο accuracy είναι:

```
❶ def define_beautiful_model():
    model = keras.models.Sequential()
    model.add(layers.Conv2D(16, (3, 3), activation="relu", padding="same", input_shape=(28,28,1)))
    model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(32, (3, 3), activation="relu", padding="same"))
    model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(32, (3, 3), activation="relu", padding="same"))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2,2)))
    model.add(layers.Dropout(0.2))
    model.add(layers.Conv2D(64, (3, 3), activation="relu", padding="same"))
    model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(64, (3, 3), activation="relu", padding="same" ))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2,2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(64,activation="relu",kernel_regularizer=keras.regularizers.l1_l2(l1=1e-5, l2=1e-4),bias_regularizer=keras.regularizers.l2(1e-4)))
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(0.3))
    model.add(layers.Dense(32,activation="relu",kernel_regularizer=keras.regularizers.l1_l2(l1=1e-5, l2=1e-4),bias_regularizer=keras.regularizers.l2(1e-4)))
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(0.25))
    model.add(layers.Dense(16,activation="relu",kernel_regularizer=keras.regularizers.l1_l2(l1=1e-5, l2=1e-4),bias_regularizer=keras.regularizers.l2(1e-4)))
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(0.2))
    model.add(layers.Dense(10,activation="softmax"))
    return model
```

Ωστόσο, παρά το γεγονός ότι στόχος είναι το μέγιστο accuracy, παράλληλα ο αριθμός των παραμέτρων δεν είναι μεγάλος. Από model summary:

```
=====
Total params: 274,298
Trainable params: 273,658
Non-trainable params: 640
```

Τέλος, όσον αφορά την εκπαίδευση και το data augmentation έχουμε:

```
data_generator = ImageDataGenerator(
    featurewise_center=False, # set input mean to 0 over the dataset
    samplewise_center=False, # set each sample mean to 0
    featurewise_std_normalization=False, # divide inputs by std of the dataset
    samplewise_std_normalization=False, # divide each input by its std
    zca_whitening=False, # dimension reduction
    rotation_range=0, # randomly rotate images in the range
    zoom_range = 0, # Randomly zoom image
    shear_range=0, # Shear angle in counter-clockwise direction in degrees
    width_shift_range=0.1, # randomly shift images horizontally
    height_shift_range=0.1, # randomly shift images vertically
    horizontal_flip=True, # randomly flip images
    vertical_flip=False, # randomly flip images
    validation_split=1/6) # Part of training used as validation

❷ checkpoints = keras.callbacks.ModelCheckpoint(filepath="/content/drive/MyDrive/Colab Notebooks/Pattern Recognition/Fashion Mnist/beautiful_net.hdf5", verbose=1, save_best_only=True)
earlystopping = keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=16, restore_best_weights=True)
logdir = os.path.join("/content/drive/MyDrive/Colab Notebooks/Pattern Recognition/logs_fashion_mnist", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=4, min_lr=1e-6)

[ ] # Compile the model
b_model.compile(loss='categorical_crossentropy',
                  optimizer=keras.optimizers.Adam(learning_rate=1e-3),
                  metrics=['acc'])
batch_s = 32

history = b_model.fit(data_generator.flow(x_train, y_train, batch_size=batch_s),
                      validation_data = data_generator.flow(x_train, y_train,batch_size=batch_s, subset='validation'),
                      callbacks=[earlystopping,checkpoints,reduce_lr,tensorboard_callback],
                      verbose=1,epochs=300, batch_size=batch_s, shuffle=True)
```

Τα αποτελέσματα είναι:

```

Epoch 00150: val_loss did not improve from 0.13338
1875/1875 [=====] - 30s 16ms/step - loss: 0.2036 - acc: 0.9452 - val_loss: 0.1341 - val_acc: 0.9636 - lr: 1.0000e-06
Epoch 00150: early stopping

```

```

▶ predictions_b=b_model.predict(x_test)
predicted_classes_b=np.argmax(predictions_b,axis=1)
y_test_values = np.argmax(y_test, axis=-1) # There are in categorical form(One hot encoded)
print(classification_report(y_test_values, predicted_classes_b, target_names=class_names,digits=5))

```

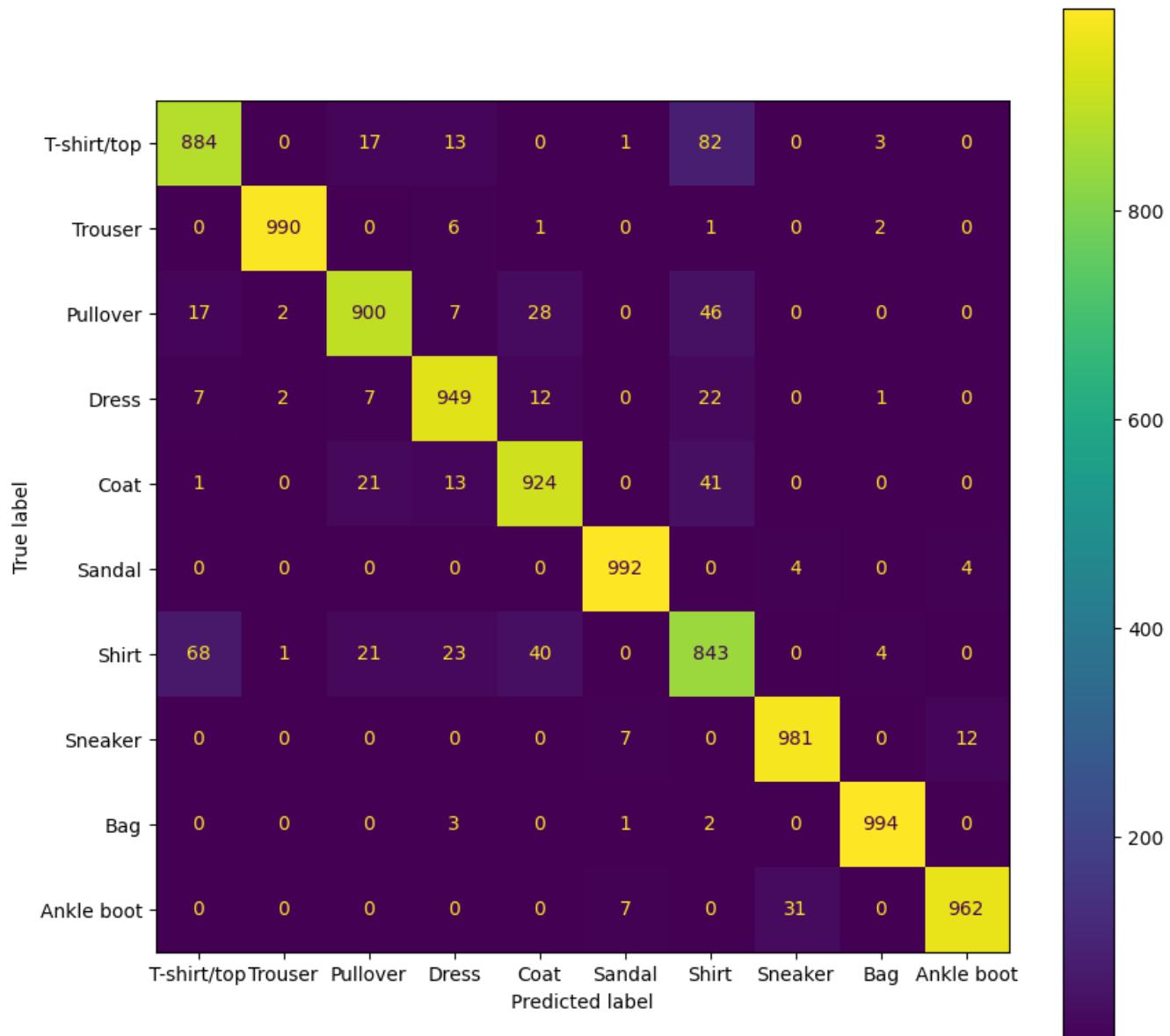
	precision	recall	f1-score	support
T-shirt/top	0.90481	0.88400	0.89428	1000
Trouser	0.99497	0.99000	0.99248	1000
Pullover	0.93168	0.90000	0.91556	1000
Dress	0.93590	0.94900	0.94240	1000
Coat	0.91940	0.92400	0.92170	1000
Sandal	0.98413	0.99200	0.98885	1000
Shirt	0.81292	0.84300	0.82769	1000
Sneaker	0.96555	0.98100	0.97321	1000
Bag	0.99004	0.99400	0.99202	1000
Ankle boot	0.98364	0.96200	0.97270	1000
accuracy		0.94190		10000
macro avg	0.94230	0.94190	0.94201	10000
weighted avg	0.94230	0.94190	0.94201	10000

Όπως παρατηρούμε, η τιμή του testing accuracy είναι **94.19%**. Όσον αφορά το training accuracy παρακάτω φαίνεται και η εποχή που είχαμε την καλύτερη επόδοση στο validation. Στην εχοχή αυτή είχαμε **94.49%** στο training accuracy. Τέλος, έχουμε και το confusion matrix.

```

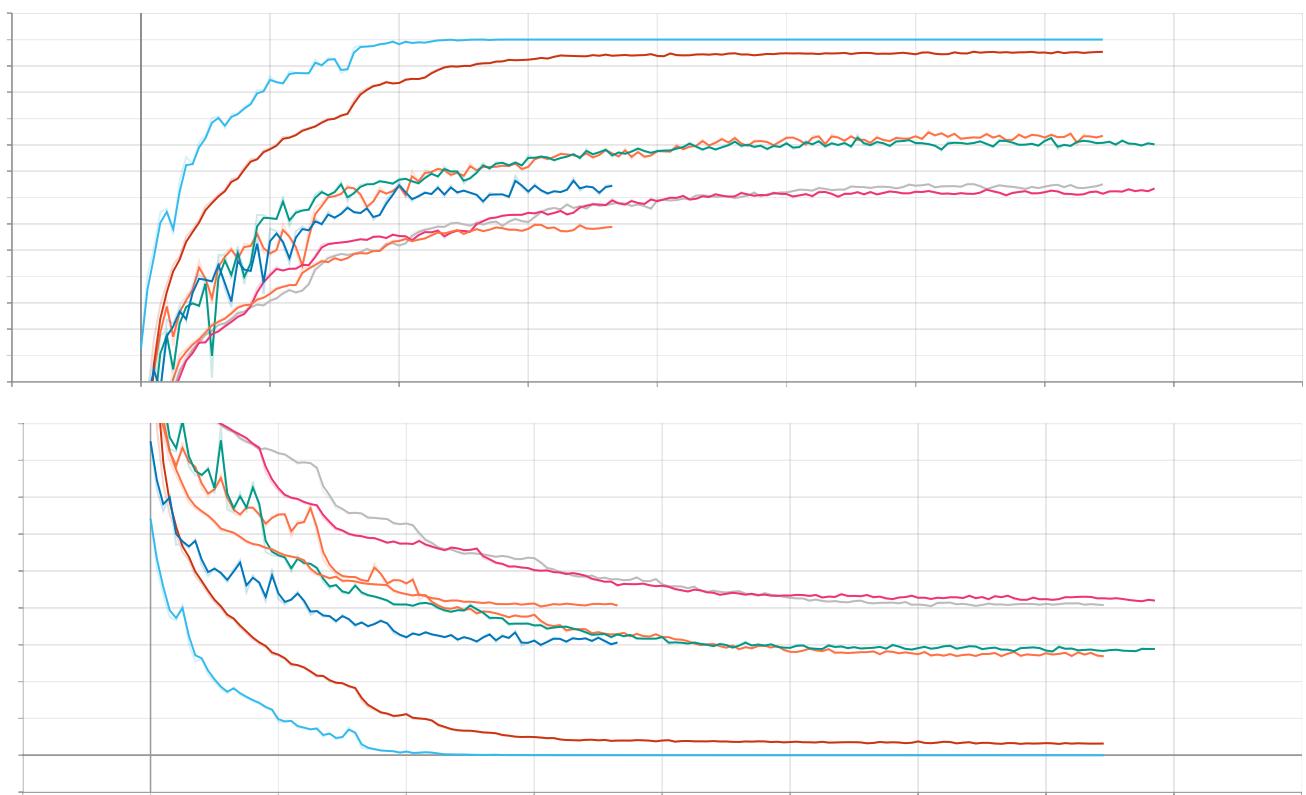
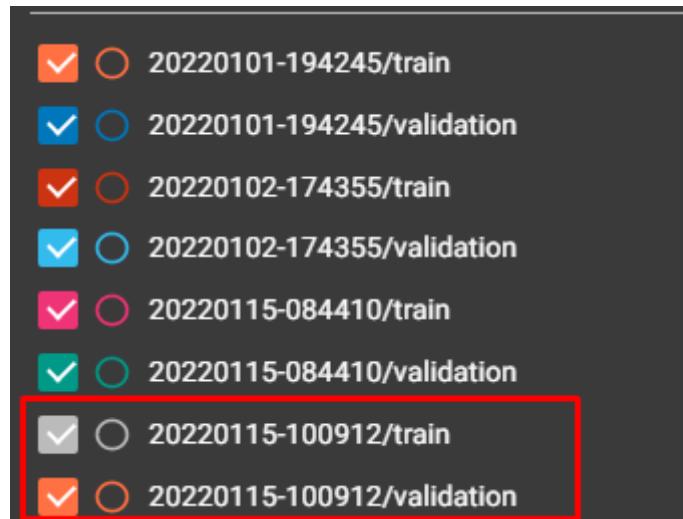
1873/1875 [=====] - ETA: 0s - loss: 0.2048 - acc: 0.9448
Epoch 00134: val_loss improved from 0.13389 to 0.13338, saving model to /content/drive/MyDrive/Colab_Notebooks/Pattern_Recognition/beautiful_net.hdf5
1875/1875 [=====] - 30s 16ms/step - loss: 0.2047 - acc: 0.9449 - val_loss: 0.1334 - val_acc: 0.9637 - lr: 1.9531e-06

```



Παρακάτω παρουσιάζονται τα training/validation accuracy και loss curves.

Το τελικό μοντέλο είναι το εξής:



Όσον αφορά τις καμπύλες με τα πολύ υψηλά accuracy και πολύ χαμηλά loss το validation set που είχε χρησιμοποιηθεί ήταν πολύ μικρό καθώς επίσης δεν είχε γίνει χρήση data augmentation.

Νευρωνικό δικής μου έμπνευσης για Cifar 10

Τα αποτελέσματα που κράτησα για το Cifar 10 στο test set κυμαίνονται από 77.77% έως 89.39%. Παρακάτω θα παρουσιαστεί το notebook και το δίκτυο για το μέγιστο accuracy. Το αρχείο του notebook ονομάζεται

Cifar_10.ipynb. Ωστόσο, στο τέλος θα υπάρχει σύνδεσμος για google drive που μπορούν να βρεθούν και τα υπόλοιπα notebooks σε υποφακέλους(καθώς έχουν όλα το ίδιο όνομα αλλά στο όνομα του φακέλου αναφέρεται και το accuracy).

Η τελική αρχιτεκτονική με στόχο το μέγιστο accuracy είναι:

```
❶ def define_my_cifar10_model():
    model = keras.models.Sequential()
    model.add(layers.Conv2D(32, (3, 3),activation="mish", padding="same",input_shape=(32,32,3)))
    model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(32, (3, 3),activation="mish", padding="same"))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(pool_size=(2, 2),strides=(2,2)))
    model.add(layers.Dropout(0.2))
    model.add(layers.Conv2D(64, (3, 3),activation="mish", padding="same"))
    model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(64, (3, 3),activation="mish", padding="same" ))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(pool_size=(2, 2),strides=(2,2)))
    model.add(layers.Conv2D(128, (3, 3),activation="mish", padding="same"))
    model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(128, (3, 3),activation="mish", padding="same" ))
    model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(256, (3, 3),activation="mish", padding="same" ))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(pool_size=(2, 2),strides=(2,2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(1024,activation="mish",kernel_regularizer=keras.regularizers.l1_l2(l1=1e-5, l2=1e-4),bias_regularizer=keras.regularizers.l2(1e-4)))
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(0.25))
    model.add(layers.Dense(256,activation="mish",kernel_regularizer=keras.regularizers.l1_l2(l1=1e-5, l2=1e-4),bias_regularizer=keras.regularizers.l2(1e-4)))
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(0.2))
    model.add(layers.Dense(10,activation="softmax"))

    return model
```

Το συγκεκριμένο μοντέλο είναι αρκετά μεγάλο δηλαδή έχει περίπου 2.5M παραμέτρους. Από model summary έχουμε:

```
=====
Total params: 2,495,786
Trainable params: 2,491,306
Non-trainable params: 4,480
=====
```

Τέλος, όσον αφορά την εκπαίδευση και το data augmentation έχουμε:

```
❷ data_generator = ImageDataGenerator(
        featurewise_center=False, # set input mean to 0 over the dataset
        samplewise_center=False, # set each sample mean to 0
        featurewise_std_normalization=False, # divide inputs by std of the dataset
        samplewise_std_normalization=False, # divide each input by its std
        zca_whitening=False, # dimesion reduction
        rotation_range=20, # randomly rotate images in the range
        zoom_range = 0, # Randomly zoom image
        shear_range=0, # Shear angle in counter-clockwise direction in degrees
        width_shift_range=0, # randomly shift images horizontally
        height_shift_range=0, # randomly shift images vertically
        horizontal_flip=True, # randomly flip images
        vertical_flip=False, # randomly flip images
        validation_split=0.2) # Part of training used as validation
```

```

❶ checkpoints = tf.keras.callbacks.ModelCheckpoint(filepath="/content/drive/MyDrive/Pattern_Recognition/my_cifar10_net.hdf5", verbose=1, save_best_only=True)
earlystopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=16, restore_best_weights=True)
logdir = os.path.join("/content/drive/MyDrive/Pattern_Recognition/logs_cifar_10", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=4, min_lr=1e-6)

[ ] # Compile the model
my_cifar10_model.compile(loss='categorical_crossentropy',
                         optimizer=keras.optimizers.Adam(learning_rate=1e-3),
                         metrics=['acc'])
batch_s = 32

# data_generator.fit(x_train)

history = my_cifar10_model.fit(data_generator.flow(x_train, y_train, batch_size=batch_s),
                                 validation_data = data_generator.flow(x_train, y_train,batch_size=batch_s, subset='validation'),
                                 callbacks=[earlystopping,checkpoints,reduce_lr,tensorboard_callback],
                                 verbose=1,epochs=250, batch_size=batch_s, shuffle=True)

```

Τα αποτελέσματα είναι τα εξής:

```

Epoch 00206: val_loss did not improve from 0.01427
1563/1563 [=====] - 42s 27ms/step - loss: 0.0370 - acc: 0.9917 - val_loss: 0.0143 - val_acc: 0.9998 - lr: 1.0000e-06
Epoch 00206: early stopping

❷ predictions_my_cifar10=my_cifar10_model.predict(x_test)
predicted_classes_my_cifar10=np.argmax(predictions_my_cifar10,axis=1)
y_test_values = np.argmax(y_test, axis=-1) # There are in categorical form(One hot encoded)
print(classification_report(y_test_values, predicted_classes_my_cifar10, target_names=class_names,digits=5))

❸
precision    recall    f1-score   support
airplane      0.91265  0.90900  0.91082     1000
automobile    0.94955  0.96000  0.95475     1000
bird          0.88066  0.85600  0.86815     1000
cat           0.79474  0.75500  0.77436     1000
deer          0.85581  0.91400  0.88395     1000
dog           0.87696  0.78400  0.82788     1000
frog          0.89098  0.94800  0.91860     1000
horse         0.91815  0.93100  0.92453     1000
ship          0.92647  0.94500  0.93564     1000
truck         0.92681  0.93700  0.93187     1000

accuracy       0.89390     10000
macro avg     0.89328  0.89390  0.89306     10000
weighted avg  0.89328  0.89390  0.89306     10000

```

Όπως παρατηρούμε, η τιμή του testing accuracy είναι **89.39%**. Όσον αφορά το training accuracy παρακάτω φαίνεται και η εποχή που είχαμε την καλύτερη επόδοση στο validation. Στην εχοχή αυτή είχαμε **99.19%** στο training accuracy.

```

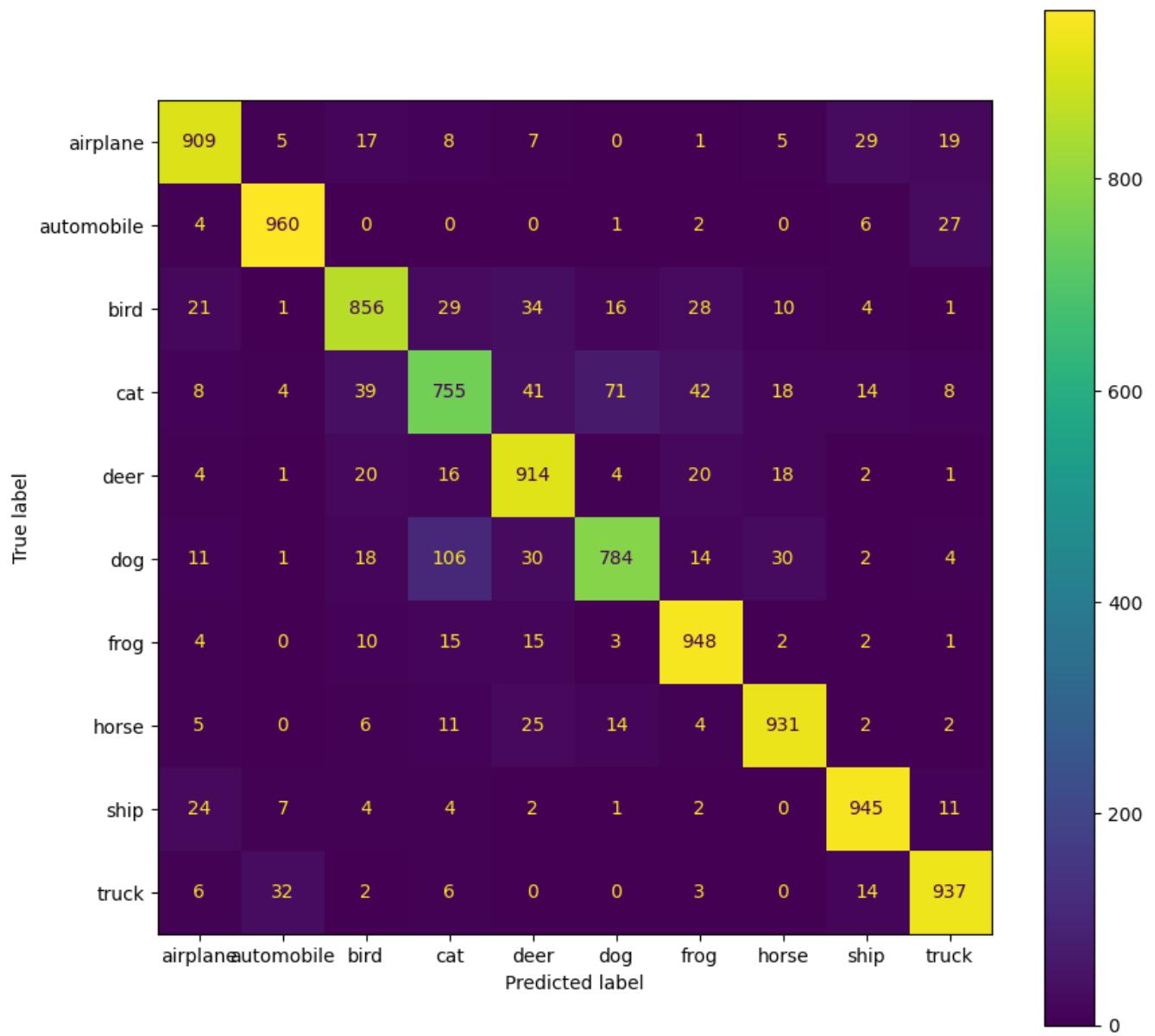
Epoch 190/250
1561/1563 [=====] - ETA: 0s - loss: 0.0370 - acc: 0.9919
Epoch 00190: val_loss improved from 0.01474 to 0.01427, saving model to /content/drive/MyDrive/Pattern_Recognition/my_cifar10_net.hdf5
1563/1563 [=====] - 42s 27ms/step - loss: 0.0370 - acc: 0.9919 - val_loss: 0.0143 - val_acc: 1.0000 - lr: 7.8125e-06

```

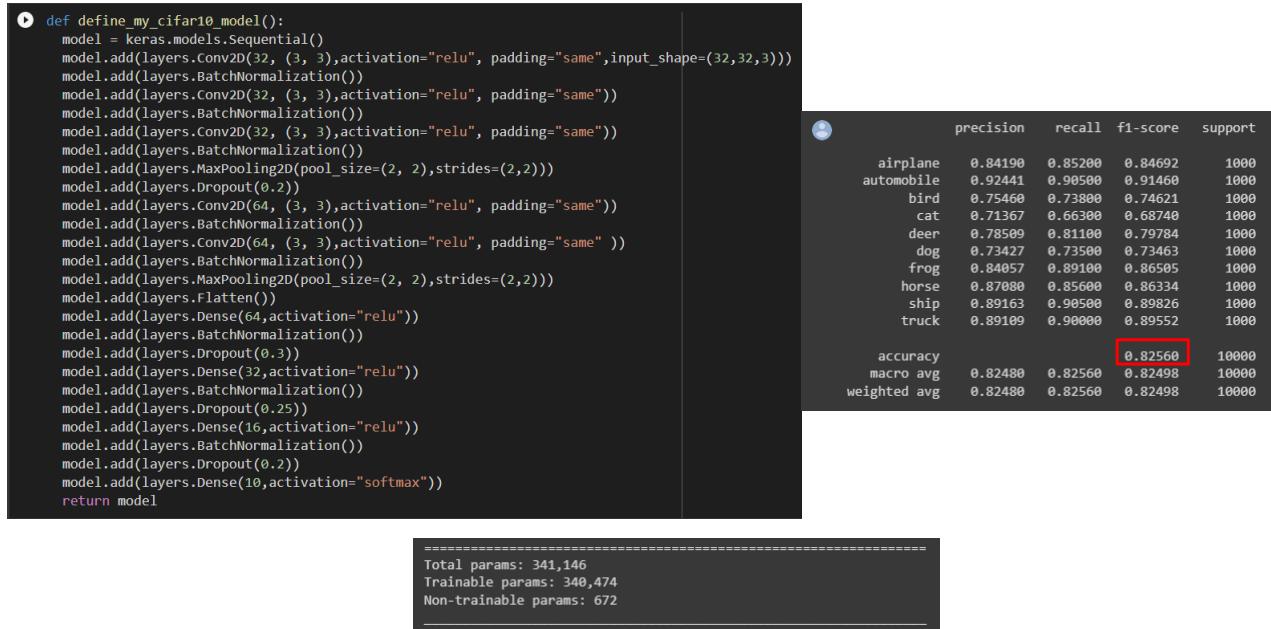
Παρατηρούμε δηλαδή στη συγκεκριμένη περίπτωση πως το δίκτυο παρόλο που έφτασε 99.19% στο training και 100% στο validation το testing accuracy έφτασε σχεδόν 90%. Αυτό, δηλώνει από τη μία τη δυσκολία του συγκεκριμένου dataset. Ειδικότερα state of the art μοντέλα που πετυχαίνουν πάνω από 90% δεν είναι απλά CNN καθώς επίσης έχουν πολλά εκατομμύρια παραμέτρους. Για παράδειγμα από τη πηγή papers with code:

1	ViT-H/14	99.50±0.06	632M	✓	An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale		2020	Transformer
2	CaIT-M-36 U224	99.4		✓	Going deeper with Image Transformers		2021	Transformer
3	CvT-W24	99.39		✓	CvT: Introducing Convolutions to Vision Transformers		2021	Transformer
4	BiT-L (ResNet)	99.37		✓	Big Transfer (BiT): General Visual Representation Learning		2019	RaNet
5	CoIT-S (384 finetune resolution)	99.1		✓	Incorporating Convolution Designs into Visual Transformers		2021	Transformer
6	AutoFormer-S 384	99.1	23M	✓	AutoFormer: Searching Transformers for Visual Recognition		2021	
7	TNT-B	99.1	65.6M	✓	Transformer in Transformer		2021	Transformer
8	DeiT-B	99.1	86M	✓	Training data-efficient image transformers & distillation through attention		2020	Transformer
9	EfficientNetV2-L	99.1	121M	✓	EfficientNetV2: Smaller Models and Faster Training		2021	EfficientNet
10	LaNet	99.03	44.1M	✗	Sample-Efficient Neural Architecture Search by Learning Action Space for Monte Carlo Tree Search		2019	

Τέλος το confusion matrix είναι:



Πέραν αυτού του νευρωνικού το αντίστοιχο που στοχεύει στο μέγιστο efficiency είναι:



Το νευρωνικό αυτό όπως βλέπουμε έχει 82.56% testing accuracy.

Νευρωνικό δική μου έμπνευσης για Cifar 100

Τα αποτελέσματα που κράτησα για το Cifar 10 στο test set κυμαίνονται από 52.13% έως 63.73%. Παρακάτω θα παρουσιαστεί το notebook και το δίκτυο για το μέγιστο accuracy. Το αρχείο του notebook ονομάζεται Cifar_100.ipynb.

Ωστόσο, στο τέλος θα υπάρχει σύνδεσμος για google drive που μπορούν να βρεθούν και τα υπόλοιπα notebooks σε υποφακέλους (καθώς έχουν όλα το ίδιο όνομα αλλά στο όνομα του φακέλου αναφέρεται και το accuracy).

Η τελική αρχιτεκτονική με στόχο το μέγιστο accuracy και ταυτόχρονα σε efficiency είναι:

```
❶ def define_my_cifar100_model():
    model = keras.models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation="relu", padding="same", input_shape=(32,32,3)))
    model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(32, (3, 3), activation="relu", padding="same"))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2,2)))
    model.add(layers.Conv2D(64, (3, 3), activation="relu", padding="same"))
    model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(64, (3, 3), activation="relu", padding="same" ))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2,2)))
    model.add(layers.Conv2D(128, (3, 3), activation="relu", padding="same" ))
    model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(128, (3, 3), activation="relu", padding="same" ))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2,2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(256,activation="relu",kernel_regularizer=keras.regularizers.l1_l2(l1=1e-5, l2=1e-4),bias_regularizer=keras.regularizers.l2(1e-4),activity_regularizer=keras.regularizers.l2(1e-5)))
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(0.25))
    model.add(layers.Dense(128,activation="relu",kernel_regularizer=keras.regularizers.l1_l2(l1=1e-5, l2=1e-4),bias_regularizer=keras.regularizers.l2(1e-4),activity_regularizer=keras.regularizers.l2(1e-5)))
    model.add(layers.BatchNormalization())
    model.add(layers.Dense(100,activation="softmax"))
    return model
```

Το συγκεκριμένο μοντέλο είναι θεωρητικά μικρό για τη δυσκολία του προβλήματος και τον αριθμό των κλάσεων δηλαδή έχει περίπου 860k παραμέτρους. Από model summary έχουμε:

```
=====
Total params: 860,676
Trainable params: 859,012
Non-trainable params: 1,664
```

```
[ ] data_generator = ImageDataGenerator(
    featurewise_center=False, # set input mean to 0 over the dataset
    samplewise_center=False, # set each sample mean to 0
    featurewise_std_normalization=False, # divide inputs by std of the dataset
    samplewise_std_normalization=False, # divide each input by its std
    zca_whitening=False, # dimension reduction
    rotation_range=20, # randomly rotate images in the range
    zoom_range = 0, # Randomly zoom image
    shear_range=0, # Shear angle in counter-clockwise direction in degrees
    width_shift_range=0, # randomly shift images horizontally
    height_shift_range=0, # randomly shift images vertically
    horizontal_flip=True, # randomly flip images
    vertical_flip=False, # randomly flip images
    validation_split=0.2) # Part of training used as alidation

❷ checkpoints = tf.keras.callbacks.ModelCheckpoint(filepath="/content/drive/MyDrive/Colab_Notebooks/Pattern_Recognition/my_cifar100_net.hdf5", verbose=1, save_best_only=True)
earlystopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=16, restore_best_weights=True)
logdir = os.path.join("/content/drive/MyDrive/Colab_Notebooks/Pattern_Recognition/logs_cifar_100", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=4, min_lr=1e-6)

[ ] # Compile the model
my_cifar100_model.compile(loss='categorical_crossentropy',
                           optimizer=keras.optimizers.Adam(learning_rate=1e-3),
                           metrics=['acc'])
batch_s = 32

history = my_cifar100_model.fit(data_generator.flow(x_train, y_train, batch_size=batch_s),
                                 validation_data = data_generator.flow(x_train, y_train,batch_size=batch_s, subset='validation'),
                                 callbacks=[earlystopping,checkpoints,reduce_lr,tensorboard_callback],
                                 verbose=1,epochs=250, batch_size=batch_s, shuffle=True)
```

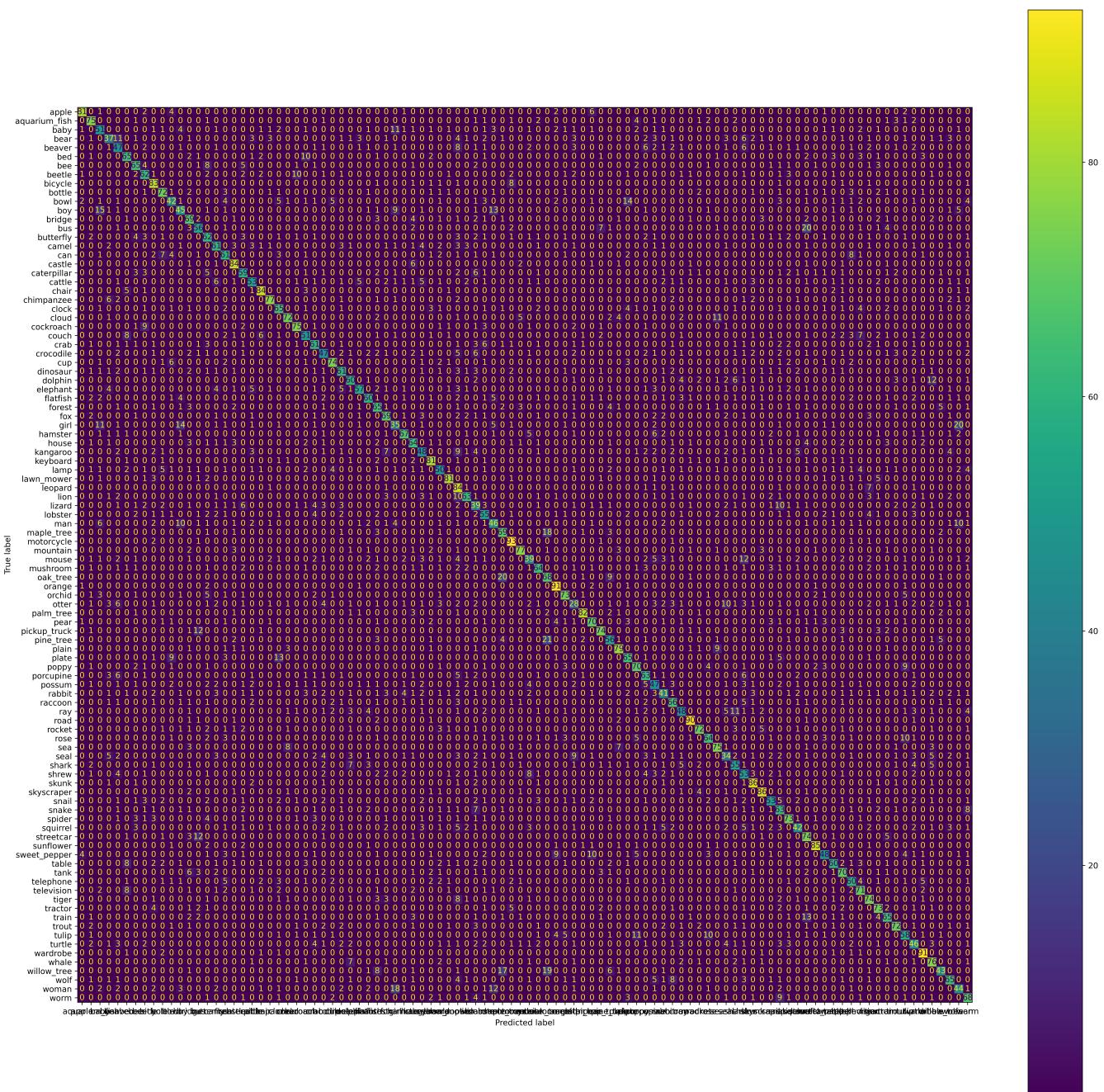
Τα αποτελέσματα είναι τα εξής:

```
Epoch 00248: val_loss improved from 0.235719 to 0.23574, saving model to /content/drive/MyDrive/Colab_Notebooks/Pattern_Recognition/my_cifar100_net.hdf5
1563/1563 [=====] - 51s 33ms/step - loss: 0.4357 - acc: 0.9180 - val_loss: 0.2357 - val_acc: 0.9896 - lr: 1.5625e-05
Epoch 249/250
1563/1563 [=====] - ETA: 0s - loss: 0.4335 - acc: 0.9185
Epoch 00249: val_loss improved from 0.23574 to 0.23316, saving model to /content/drive/MyDrive/Colab_Notebooks/Pattern_Recognition/my_cifar100_net.hdf5
1563/1563 [=====] - 51s 32ms/step - loss: 0.4335 - acc: 0.9185 - val_loss: 0.2332 - val_acc: 0.9905 - lr: 1.5625e-05
Epoch 250/250
1563/1563 [=====] - ETA: 0s - loss: 0.4295 - acc: 0.9206
Epoch 00250: val_loss did not improve from 0.23316
1563/1563 [=====] - 50s 32ms/step - loss: 0.4295 - acc: 0.9206 - val_loss: 0.2413 - val_acc: 0.9873 - lr: 1.5625e-05
```

Όπως παρατηρούμε το μοντέλο έφτασε το μέγιστο αριθμό εποχών χωρίς να σταματήσει από early stopping πράγμα που σημαίνει ότι πιθανών να μπορούσε να συνεχίσει η εκπαίδευση. Όπως παρατηρούμε, η τιμή του testing accuracy είναι **63.73%**. Όσον αφορά το training accuracy παρακάτω φαίνεται και η εποχή που είχαμε την καλύτερη επόδοση στο validation. Στην εχοχή αυτή είχαμε **91.85%** στο training accuracy.

	precision	recall	f1-score	support	fox	0.68317	0.69000	0.68657	100
apple	0.89011	0.81000	0.84817	100	girl	0.41667	0.35000	0.38043	100
aquarium_fish	0.72115	0.75000	0.73529	100	hamster	0.77907	0.67000	0.72043	100
baby	0.50000	0.51000	0.50495	100	house	0.61538	0.64000	0.62745	100
bear	0.51389	0.37000	0.43023	100	kangaroo	0.55814	0.48000	0.51613	100
beaver	0.43119	0.47000	0.44976	100	keyboard	0.74312	0.81000	0.77512	100
bed	0.61905	0.65000	0.63415	100	lamp	0.54945	0.50000	0.52356	100
bee	0.67010	0.65000	0.65990	100	lawn_mower	0.81818	0.81000	0.81407	100
beetle	0.60194	0.62000	0.61084	100	leopard	0.41791	0.84000	0.55814	100
bicycle	0.69167	0.83000	0.75455	100	lion	0.73256	0.63000	0.67742	100
bottle	0.69903	0.72000	0.70936	100	lizard	0.33051	0.39000	0.35780	100
bowl	0.53846	0.42000	0.47191	100	lobster	0.52381	0.55000	0.53659	100
boy	0.48913	0.45000	0.46875	100	man	0.51111	0.46000	0.48421	100
bridge	0.56557	0.69000	0.62162	100	maple_tree	0.57983	0.69000	0.63014	100
bus	0.56000	0.56000	0.56000	100	motorcycle	0.74400	0.93000	0.82667	100
butterfly	0.51240	0.62000	0.56109	100	mountain	0.81053	0.77000	0.78974	100
camel	0.62245	0.61000	0.61616	100	mouse	0.52000	0.39000	0.44571	100
can	0.59804	0.61000	0.60396	100	mushroom	0.69565	0.64000	0.66667	100
castle	0.75676	0.84000	0.79621	100	oak_tree	0.52713	0.68000	0.59389	100
caterpillar	0.60825	0.59000	0.59898	100	orange	0.76471	0.91000	0.83105	100
cattle	0.63095	0.53000	0.57609	100	orchid	0.75258	0.73000	0.74112	100
chair	0.81553	0.84000	0.82759	100	otter	0.40000	0.28000	0.32941	100
chimpanzee	0.84615	0.77000	0.80628	100	palm_tree	0.90110	0.82000	0.85864	100
clock	0.60185	0.65000	0.62500	100	pear	0.70000	0.70000	0.70000	100
cloud	0.88000	0.72000	0.75789	100	pickup_truck	0.76289	0.74000	0.75127	100
cockroach	0.77320	0.75000	0.76142	100	pine_tree	0.58947	0.56000	0.57436	100
couch	0.58621	0.51000	0.54545	100	plain	0.77451	0.79000	0.78218	100
crab	0.62887	0.61000	0.61929	100	plate	0.61905	0.65000	0.63415	100
crocodile	0.52222	0.47000	0.49474	100	poppy	0.67308	0.70000	0.68627	100
cup	0.79570	0.74000	0.76684	100	porcupine	0.58879	0.63000	0.60870	100
dinosaur	0.64211	0.61000	0.62564	100	possum	0.47475	0.47000	0.47236	100
dolphin	0.63830	0.60000	0.61856	100	rabbit	0.51250	0.41000	0.45556	100
elephant	0.78082	0.57000	0.65896	100	raccoon	0.64078	0.66000	0.65025	100
flattfish	0.68966	0.60000	0.64171	100	ray	0.60759	0.48000	0.53631	100
forest	0.52846	0.65000	0.58296	100	road	0.86538	0.90000	0.88235	100
rocket	0.79121	0.72000	0.75393	100					
rose	0.70330	0.64000	0.67016	100					
sea	0.71429	0.75000	0.73171	100					
seal	0.41975	0.34000	0.37569	100					
shark	0.59783	0.55000	0.57292	100					
shrew	0.43089	0.53000	0.47534	100					
skunk	0.75439	0.86000	0.80374	100					
skyscraper	0.78899	0.86000	0.82297	100					
snail	0.65432	0.53000	0.58564	100					
snake	0.46324	0.63000	0.53390	100					
spider	0.71569	0.73000	0.72277	100					
squirrel	0.60000	0.42000	0.49412	100					
streetcar	0.52857	0.74000	0.61667	100					
sunflower	0.88542	0.85000	0.86735	100					
sweet_pepper	0.68571	0.48000	0.56471	100					
table	0.69767	0.60000	0.64516	100					
tank	0.66667	0.70000	0.68293	100					
telephone	0.61856	0.60000	0.60914	100					
television	0.65138	0.71000	0.67943	100					
tiger	0.64348	0.74000	0.68837	100					
tractor	0.66364	0.73000	0.69524	100					
train	0.65657	0.65000	0.65327	100					
trout	0.76596	0.72000	0.74227	100					
tulip	0.58586	0.58000	0.58291	100					
turtle	0.48936	0.46000	0.47423	100					
wardrobe	0.79825	0.91000	0.85047	100					
whale	0.61789	0.76000	0.68161	100					
willow_tree	0.68254	0.43000	0.52761	100					
wolf	0.68421	0.65000	0.66667	100					
woman	0.48352	0.44000	0.46073	100					
worm	0.57143	0.68000	0.62100	100					
accuracy			0.63730	10000					
macro avg	0.64063	0.63730	0.63502	10000					
weighted avg	0.64063	0.63730	0.63502	10000					

To confusion matrix είναι:



Google Drive Link

Μέσω του παρακάτω link μπορείτε να δείτε το σύνολο των δοκιμών στο google drive: <https://drive.google.com/drive/folders/11AYG72MVVT4tWDpB81JGBSS1YyGgK1Jk?usp=sharing>

