Deep Learning - Εργασια

Ρούσος Σταμάτης

2 Φεβρουαρίου 2026

1 Εισαγωγή

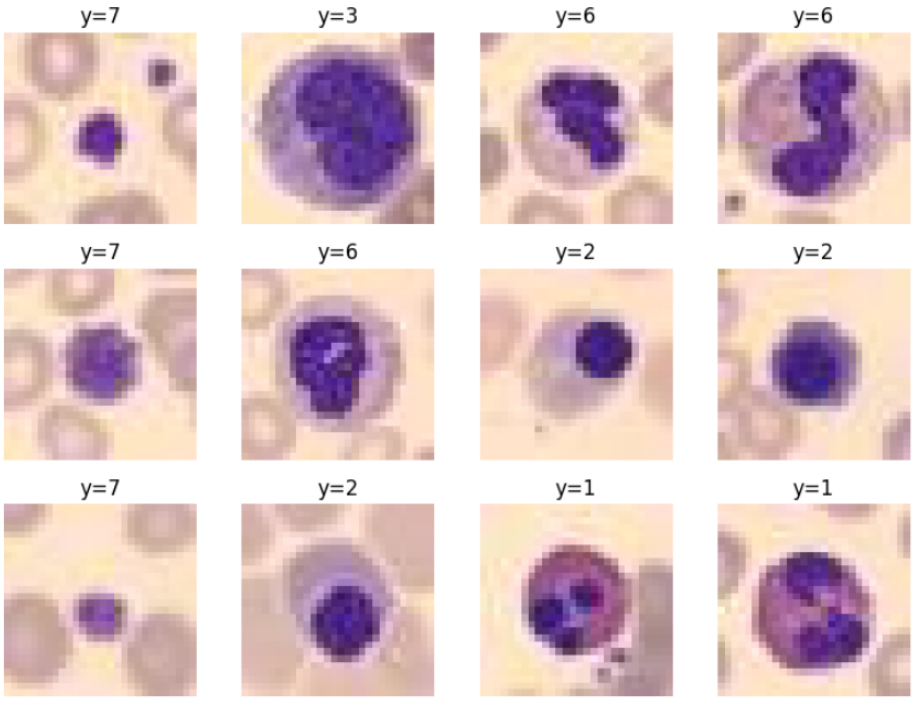
Στην παρούσα εργασία θα ασχοληθούμε με τη βαθιά μάθηση εκπαιδεύοντας νευρωνικά δίκτυα. Η εργασία χωρίζεται σε 3 μέρη:

- CNN με διαφορους πειραματισμους τεχνικων και υπερπαραμετρων
- Transfer Learning στο ResNet18
- Transfer Learning στον DeIT transformer

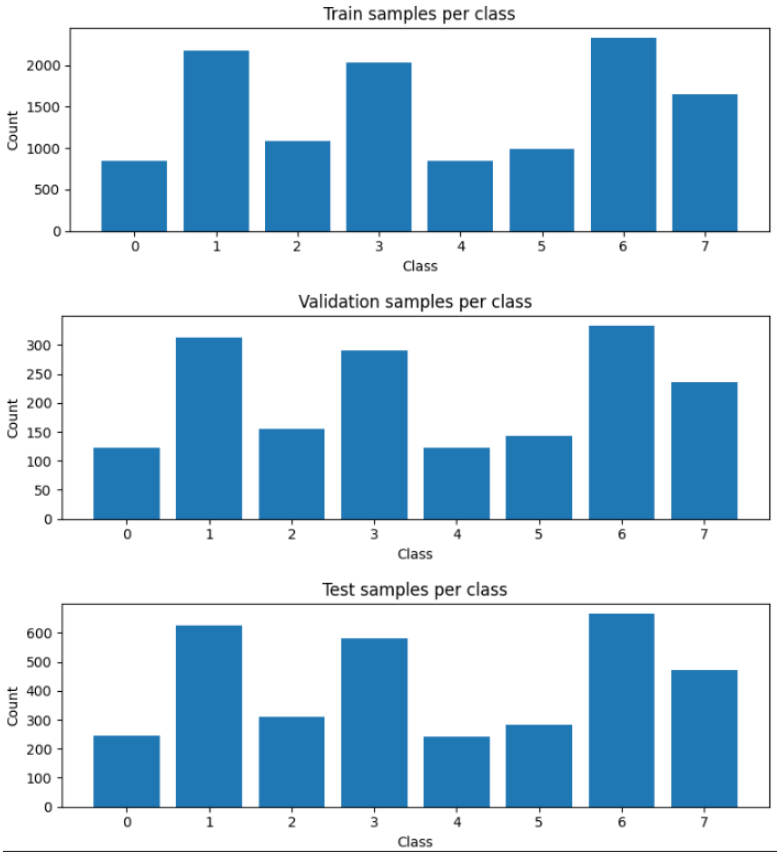
Το Dataset στο οποίο έγιναν οι μελέτες είναι το BloodMNIST της MedMnist. Το Dataset περιέχει συνολικά 17.092 εικόνες, οι οποίες χωρίζονται σε 11.959 training images, 1.712 validation images, και 3.421 test images, μεγέθους 28x28x3, όπου 3 τα κανάλια RGB. Οι εικόνες του Dataset είναι blood smears που λήφθηκαν από μικροσκόπιο και δείχνουν ένα κύτταρο αίματος το οποίο θέλουμε να βρούμε σε τι κατηγορία ανήκει. Οι κατηγορίες του είναι:

1. Basophil
2. Eosinophil
3. Erythroblast
4. Immature granulocytes
5. Lymphocyte
6. Monocyte
7. Neutrophil
8. platelet

Παρακάτω βλέπουμε μερικές από τις εικόνες και τις κατηγορίες τους:



Το Dataset είναι imbalanced, δηλαδή οι εικόνες δεν είναι μοιρασμένες ισόποσα σε κάθε κλάση:



2 Part 1 - CNN

2.1 Λειτουργία Convolutional Neural Network (CNN)

Τα CNNs δεν περιέχουν ακριβώς νευρώνες όπως ένα fully connected (Dense) NN, αλλά από **φίλτρα (kernels)** που σαρώνουν την εικόνα και εξάγουν χαρακτηριστικά.

Κάθε φίλτρο είναι ένας μικρός πίνακας διαστάσεων 3×3 (kernel size $= 3$), ο οποίος σαρώνει κάθε κανάλι της εικόνας εισόδου. Σε κάθε θέση, το φίλτρο εκτελεί **συνέλιξη** με το 3×3 παράθυρο της εικόνας το οποίο βλέπει εκείνη τη στιγμή. Το αποτέλεσμα της συνέλιξης είναι μία νέα "εικόνα", το οποίο συνήθως λέγεται feature map. Κάθε feature map περιέχει τις θέσεις των μοτίβων που ανιχνεύτηκαν μέσω της συνέλιξης. Στα πρώτα συνελικτικά επίπεδα, το μοντέλο μαθαίνει βασικά σχήματα όπως οριζόντιες/κατακόρυφες γραμμές, και μέσω των μη γραμμικών ενεργοποιήσεων που του βάζουμε, μαθαίνει πιο πολύπλοκα σχήματα όσο προχωράμε σε βάθος, όπως κύκλοι, τρίγωνα κ.λπ. Έτσι, ο τρόπος που τελικά κατηγοριοποιεί, είναι ανιχνεύοντας μοτίβα που υπάρχουν σε πολλές εικόνες κάθε κατηγορίας, και όταν τα ανιχνεύει, οι συναρτήσεις ενεργοποίησης δίνουν πολύ μεγάλες τιμές.

Κάθε feature map στην έξοδο ενός φίλτρου έχει το ίδιο μέγεθος με την αρχική εικόνα, λόγω του padding $= 1$ που έθεσα (και stride $= 1$ by default). Αυτό προκύπτει ως εξής:

- Το αποτέλεσμα της συνέλιξης μιας εικόνας $N \times N$ με ένα φίλτρο $F \times F$ που μετακινείται κατά μία στήλη δεξιά σε κάθε βήμα έχει μέγεθος $(N - F + 1) \times (N - F + 1)$.

$$N_{\text{out}} = N - F + 1 + 2P$$

- Έστω 2P οι γραμμές/στήλες μηδενικών που προσθέτουμε στην εικόνα εισόδου. Τοτε οι διαστάσεις της γίνονται $(N + 2P) \times (N + 2P)$

- Αυτή η εικόνα συνελίσσεται με το φίλτρο, και παράγει feature map μεγέθους $N_{out} \times N_{out} = (N + 2P - F + 1) \times (N + 2P - F + 1)$ ή για $F = 3$: $(N + 2P - 2) \times (N + 2P - 2)$

- Άρα, για να ισχύει $N_{out} = N$, πρέπει $P = 1$ Έτσι, τα feature maps είναι ίδιου μεγέθους με την εικόνα εισόδου.

Μετά τη συνέλιξη εφαρμόζεται ένα **MaxPooling2D** layer με kernel size $= 2$. Το MaxPooling σαρώνει το feature map με ένα παράθυρο 2×2 και κρατάει στην

έξοδο τη μέγιστη τιμή από τα τέσσερα στοιχεία του παραθύρου. Έτσι, μειώνει τις διαστάσεις του feature map στο μισό. Αυτό κάνει την εκπαίδευση πιο γρήγορη όσο προχωράμε βαθύτερα στο μοντέλο, επειδή έχουμε λιγότερες ενεργοποιήσεις, ενώ ταυτόχρονα διατηρείται η κυρίαρχη πληροφορία. Όμως χάνει πληροφορία που ίσως να είναι σημαντική. Υπάρχουν και άλλες τεχνικές Pooling όπως Average Pooling όπου κρατάμε το μέσο όρο των στοιχείων του feature map ανά παράθυρο.

Η συνάρτηση ενεργοποίησης που χρησιμοποιήθηκε στο CNN είναι η ReLU, η οποία ορίζεται ως:

$$\text{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Η ReLU είναι η πιο συνηθισμένη επιλογή σήμερα και δίνει πολύ καλά αποτελέσματα.

2.2 Εκπαίδευση με Backpropagation

Η εκπαίδευση του CNN γίνεται με τον αλγόριθμο **backpropagation**. Το backpropagation εκτελεί ένα backward pass με σκοπό τον υπολογισμό των παραγώγων (gradients) $\frac{\partial E}{\partial w_{ij}}$ για κάθε βάρος w_{ij}, όπου E είναι το συνολικό σφάλμα.

Έστω ότι χρησιμοποιούμε Mean Squared Error (MSE):

$$E = \frac{1}{2} \sum_k e_k^2, \quad e_k = d_k - y_k$$

όπου d_k είναι το target και y_k η έξοδος του νευρώνα k στο output layer.

Για έναν νευρώνα k στο output layer ισχύει:

$$\frac{\partial E}{\partial w_{ik}} = \frac{\partial E}{\partial e_k} \frac{\partial e_k}{\partial y_k} \frac{\partial y_k}{\partial u_k} \frac{\partial u_k}{\partial w_{ik}}$$

όπου w_{ik} το βάρος που συνδέει το νευρώνα i με τον k, u_k η είσοδος του νευρώνα k, και $y_k = f(u_k)$, $u_k = \sum_i w_{ik} y_i$. Έχουμε:

$$\frac{\partial E}{\partial e_k} = e_k, \quad \frac{\partial e_k}{\partial y_k} = -1, \quad \frac{\partial y_k}{\partial u_k} = f'(u_k), \quad \frac{\partial u_k}{\partial w_{ik}} = y_i$$

Άρα:

$$\frac{\partial E}{\partial w_{ik}} = -e_k f'(u_k) y_i$$

Ορίζουμε:

$$\delta_k = -\frac{\partial E}{\partial u_k} = e_k f'(u_k)$$

την "κλίση" του σφάλματος, που εκφράζει πόσο επηρεάζει το σφάλμα μια μικρή αλλαγή στην είσοδο του νευρώνα k. Αυτό μπορεί να υπολογιστεί άμεσα από το μοντέλο αφού ξέρουμε τα σφάλματα e_k, τις συναρτήσεις ενεργοποίησης και τις εισόδους u_k από το forward pass. Οπότε έχουμε:

$$\frac{\partial E}{\partial w_{ik}} = \delta_k y_i$$

Επομένως, γνωρίζοντας τις κλίσεις δ_k του output layer, ξέρουμε τα gradients για τα τελευταία βάρη w_{ik}.

Αφού υπολογιστούν τα δ_k για όλους τους νευρώνες του τελευταίου επιπέδου, προχωράμε στο προτελευταίο επίπεδο. Οι νευρώνες σε αυτό, δεν έχουν targets d_i οπότε δεν ορίζονται τα σφάλματα e_i. Η έξοδος του νευρώνα j (y_j) επηρεάζει την είσοδο όλων των νευρώνων k στο επόμενο επίπεδο, επομένως από chain rule έχουμε:

$$\frac{\partial E}{\partial y_i} = \sum_k \frac{\partial E}{\partial u_k}\frac{\partial u_k}{\partial y_i} = \sum_k \delta_k w_{ik}.$$

Άρα τελικά:

$$\frac{\partial E}{\partial w_{ji}} = \sum_k \delta_k w_{ik} f'(u_i) y_j.$$

Ορίζουμε:

$$\delta_i = -\frac{\partial E}{\partial u_i} = -f'(u_i)\sum_k \delta_k w_{ik}$$

οπότε:

$$\frac{\partial E}{\partial w_{ji}} = -\delta_i y_j.$$

Επομένως βλέπουμε ότι, αν γνωρίζουμε τις κλίσεις δ_j όλων των νευρώνων στο επίπεδο, γνωρίζουμε και τα gradients. Συνολικά:

Το backward pass υπολογίζει αρχικά τις κλίσεις

$$\delta_k = (d_k - y_k) f'(u_k)$$

του output layer, προχωράει στο επόμενο layer και υπολογίζει τα

$$\delta_i = -f'(u_i)\sum_k \delta_k w_{ik},$$

κ.ο.κ. Αφού υπολογίσει όλα τα δ_i, ενημερώνει τα βάρη:

$$w'_{ij} = w_{ij} - \eta\frac{\partial E}{\partial w_{ij}} = w_{ij} + \eta\delta_j y_i.$$

όπου η ο ρυθμός εκμάθησης (learning rate).

2.3 Classification και Cross Entropy Loss

Στα προβλήματα ταξινόμησης εικόνων, ως Loss Function χρησιμοποιείται συνήθως η **Cross Entropy Loss**, η οποία ταιριάζει πολύ με το classification, καθώς δέχεται ως είσοδο πιθανότητες. Το output layer αποτελείται από τόσους νευρώνες όσες και οι κλάσεις του dataset (στην περίπτωσή μας 8).

Κάθε νευρώνας στο output layer παράγει μία έξοδο που ονομάζεται **logit**. Τα logits περνάνε από τη softmax:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

η οποία μετατρέπει τα logits σε πιθανότητες στο διάστημα $[0, 1]$ που αθροίζουν στο 1.

Για παράδειγμα:

$$\text{logits} = [2.56,\ -1.23,\ 9.5, \dots]$$

$$\text{softmax} = [0.03,\ 0.02,\ 0.9, \dots]$$

Η κατηγορία της εικόνας είναι αυτή με τη μεγαλύτερη πιθανότητα (label $= 2$).

Το όλο πρόβλημα λοιπόν που θέλουμε να λύσουμε, είναι προσέγγιση συνάρτησης που θα παράγει 8 εξόδους:

$$f(x) = [f_1(x), f_2(x), \dots, f_8(x)]$$

καθεμία από τις οποίες θα αντιστοιχεί σε ένα logit. Το πετυχαίνουμε ψάχνοντας τα βάρη που ελαχιστοποιούν τη loss function κατά την εκπαίδευση. Επειδή η ελαχιστοποίηση αυτή γίνεται πάνω στο training dataset, για διάφορους λόγους (εύκολο dataset, πολύπλοκο μοντέλο, κ.λπ) μπορεί να οδηγηθούμε σε υπερεκπαίδευση, δηλαδή το μοντέλο να μάθει πολύ καλά τα δεδομένα εκπαίδευσης (παπαγαλίζει), αλλά να μην αποδίδει καλά σε διαφορετικά άγνωστα δεδομένα. Υπάρχουν διάφορες τεχνικές regularization που πολεμούν αυτό το φαινόμενο. Μερικές από αυτές που εφάρμοσα στην εργασία είναι:

- Dropout
- Weight Decay (L2)
- Early Stopping
- Data Augmentation

Επιπλέον, εφάρμοσα τις τεχνικές:

- Batch Normalization

- Layer Normalization

Αυτές εστιάζουν πιο πολύ στην καλύτερη εκπαίδευση. Ας δούμε σύντομα πως δουλεύει η καθεμιά.

2.3.1 Batch Normalization

Η Batch Normalization χρησιμοποιείται για να κανονικοποιεί τις ενεργοποιήσεις ενός layer κατά τη διάρκεια της εκπαίδευσης. Συγκεκριμένα, για κάθε feature υπολογίζουμε τον μέσο όρο και τη διασπορά πάνω στο mini-batch.

Για ένα mini-batch B $= \{x_1, x_2, \ldots, x_m\}$ έχουμε:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i, \quad \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

η μέση τιμή και η διασπορά του batch αντίστοιχα, και η κανονικοποίηση που κάνει είναι:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Έτσι τα $\hat{x}_i$ έχουν μέση τιμή 0 και διασπορά 1.

Προσοχη, η κανονικοποίηση γίνεται για κάθε feature της εικόνας x_i, δηλαδή τα μ_B και σ_B^2 είναι πίνακες με στοιχεία τους μέσους όρους και τις διασπορές των δειγμάτων του batch για κάθε feature. Στη συνέχεια κάνει το μετασχηματισμό:

$$y_i = \gamma\hat{x}_i + \beta$$

όπου γ και β είναι εκπαιδεύσιμες παράμετροι. Οι παράμετροι αυτοί, αντιστοιχούν στη μέση τιμή και τη τυπική απόκλιση του y_i αφού:

$$E[y_i] = E[\gamma\hat{x}_i + \beta] = \gamma E[\hat{x}_i] + E[\beta] = \beta$$

και,

$$\sigma_{y_i}^2 = E((y_i - E[y_i])^2) = [(\gamma\hat{x}_i + \beta - \beta)^2] = E[(\gamma\hat{x}_i)^2] = \gamma^2$$

Με αυτόν τον τρόπο, οι ενεργοποιήσεις διατηρούν σταθερή κατανομή κατά την εκπαίδευση, κάτι που κάνει τη σύγκλιση πιο γρήγορη και σταθερή. Επίσης, μέσω των β,γμπορεί να μαθαίνει ποια κατανομή το βολεύει από epoch σε epoch. Χωρίς αυτά, η κατανομή των $\hat{x}_i$ θα ήταν πάντα ίδια (μ=0, σ=1).

Σημειώνω ότι το Batch Norm εφαρμόζεται στα feature maps πριν μπουν στη ReLU. Δηλαδή η ReLU δέχεται ως εισόδους τα y_i.

2.3.2 Layer Normalization

Η Layer Normalization είναι παρόμοια με τη Batch Normalization, αλλά αντί να κανονικοποιεί πάνω στο batch, κανονικοποιεί πάνω στα χαρακτηριστικά του ίδιου δείγματος.

Για ένα δείγμα με H χαρακτηριστικά $x = (x_1, x_2, ..., x_H)$ έχουμε:

$$\mu = \frac{1}{H}\sum_{j=1}^{H} x_j, \quad \sigma^2 = \frac{1}{H}\sum_{j=1}^{H}(x_j - \mu)^2$$

και:

$$\hat{x}_j = \frac{x_j - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad y_j = \gamma\hat{x}_j + \beta$$

Η Layer Normalization δεν εξαρτάται από το batch size, κάτι που την κάνει ιδιαίτερα χρήσιμη σε αρχιτεκτονικές όπως οι Transformers που δουλεύουν με μικρά batches.

2.3.3 Dropout

Το Dropout είναι μια τεχνική regularization όπου, κατά τη διάρκεια της εκπαίδευσης, κάθε νευρώνας μηδενίζεται τυχαία με πιθανότητα p.

Για την έξοδο ενός νευρώνα y_i ορίζουμε μια τυχαία μεταβλητή:

$$r_i \sim \text{Bernoulli}(1 - p)$$

και η έξοδος μετά το Dropout είναι:

$$\tilde{y}_i = \frac{r_i}{1 - p} \cdot y_i$$

Έτσι, σε κάθε forward pass, το δίκτυο χρησιμοποιεί ένα διαφορετικό υποσύνολο νευρώνων. Αυτό αποτρέπει το overfitting, γιατί το μοντέλο δεν μπορεί να βασίζεται υπερβολικά σε συγκεκριμένους νευρώνες. Κατά το inference, το Dropout απενεργοποιείται.

2.3.4 Weight Decay (L2 Regularization)

Το Weight Decay προσθέτει έναν όρο κανονικοποίησης στη συνάρτηση σφάλματος, ο οποίος τιμωρεί μεγάλα βάρη.

Αν E είναι το αρχικό σφάλμα, τότε το συνολικό σφάλμα γίνεται:

$$E_{\text{total}} = E + \lambda \sum_i w_i^2$$

Κατά την ενημέρωση των βαρών έχουμε:

$$w_i' = w_i - \eta \left(\frac{\partial E}{\partial w_i} + \lambda w_i \right)$$

Με αυτόν τον τρόπο, όσο πάνε να μεγαλώσουν τα βάρη, μεγαλώνει και το συνολικό σφάλμα. Έτσι, το μοντέλο σπρώχνει τα βάρη σε μικρές τιμές, κάτι που βοηθά στη γενίκευση του μοντέλου. Όσο πιο μεγάλο το λ, τόσο πιο μεγάλη η ποινή.

2.3.5 Early Stopping

Το Early Stopping είναι μια τεχνική regularization όπου η εκπαίδευση σταματάει όταν κάποιο metric (εγώ έβαλα το validation loss γιατί είναι συνήθως πιο σταθερό σαν metric) δε βελτιώνεται. Αν για έναν αριθμό epochs (patience) το validation loss δεν μειώνεται, τότε η εκπαίδευση διακόπτεται ώστε να αποφευχθεί το overfitting.

2.3.6 Data Augmentation

Το Data Augmentation αυξάνει τεχνητά το μέγεθος του dataset κάνοντας τυχαίους μετασχηματισμούς, όπως περιστροφές, flips κ.λπ.

Με αυτόν τον τρόπο, το μοντέλο βλέπει διαφορετικές εκδοχές των ίδιων δεδομένων. Συνήθως βοηθάει στη γενίκευση.

3 Πειραματικά Αποτελέσματα

- Σε όλα τα μοντέλα το patience του Early Stopping ήταν μεταξύ 5-10
- Ο optimizer ήταν πάντα Adam
- Το batch size ήταν για όλα τα μοντέλα 64
- Το early stopping το ρύθμισα να κάνει monitor το validation loss, και να επαναφέρει τα βάρη εκείνης της εποχής που είχε το καλύτερο. Επομένως, θεωρώ αυτό ως τελικό μοντέλο και τα υπόλοιπα metrics είναι εκείνης της εποχής, παρόλο που μπορεί στις επόμενες εποχές, πριν σταματήσει, να πέτυχε καλύτερο training/validation accuracy.

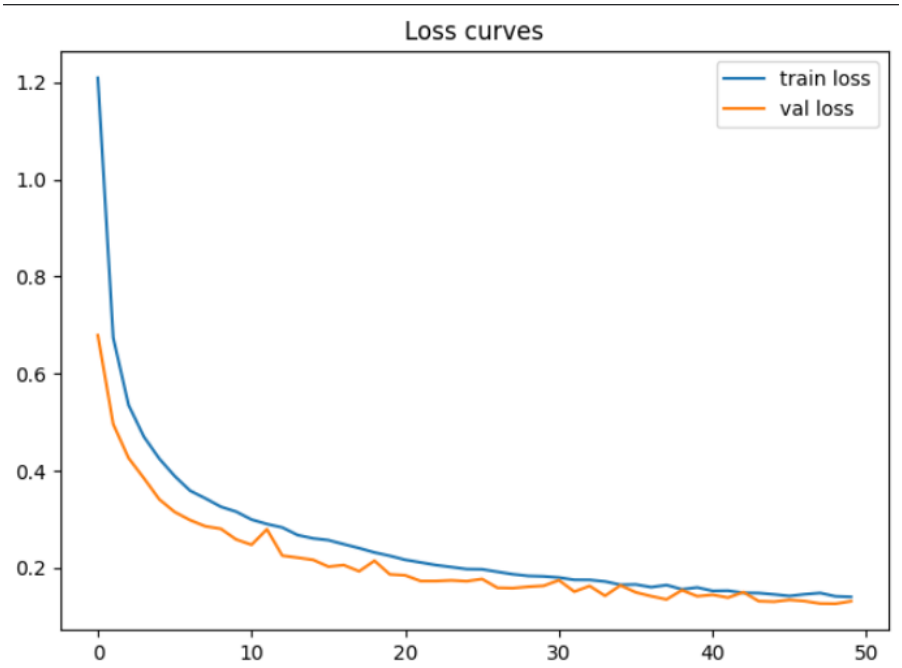
1) BN - Batch Normalization, 2) Dr: Dropout rate, 3) WD(λ): Weight Decay, 4) lr: learning rate, 5) ep: η εποχή του καλύτερου μοντέλου, 6) Tr Acc: training

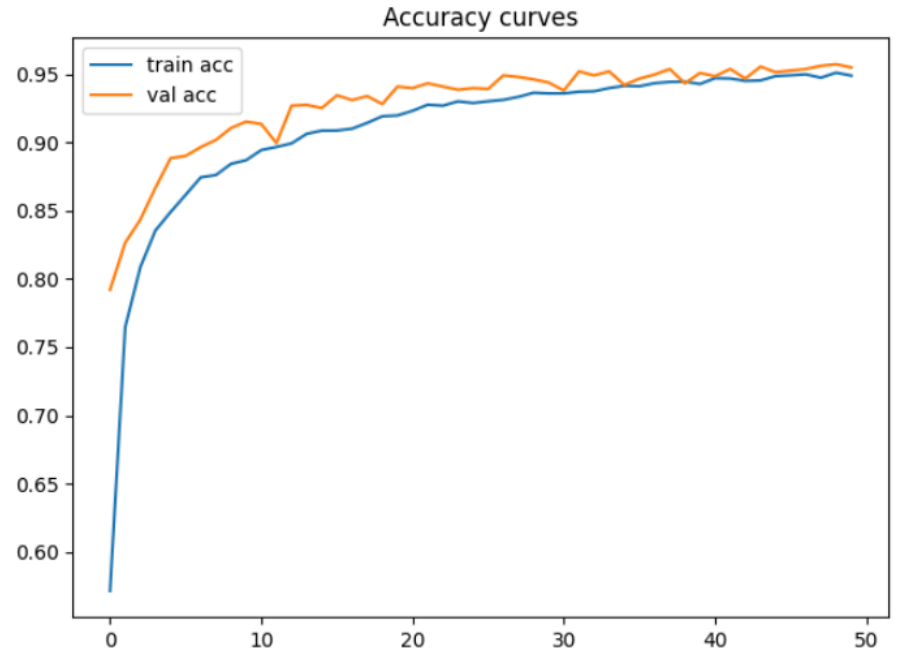
accuracy, 7) VL: best validation loss, 8) Val Acc: validation accuracy, 9) LN: Layer Normalization

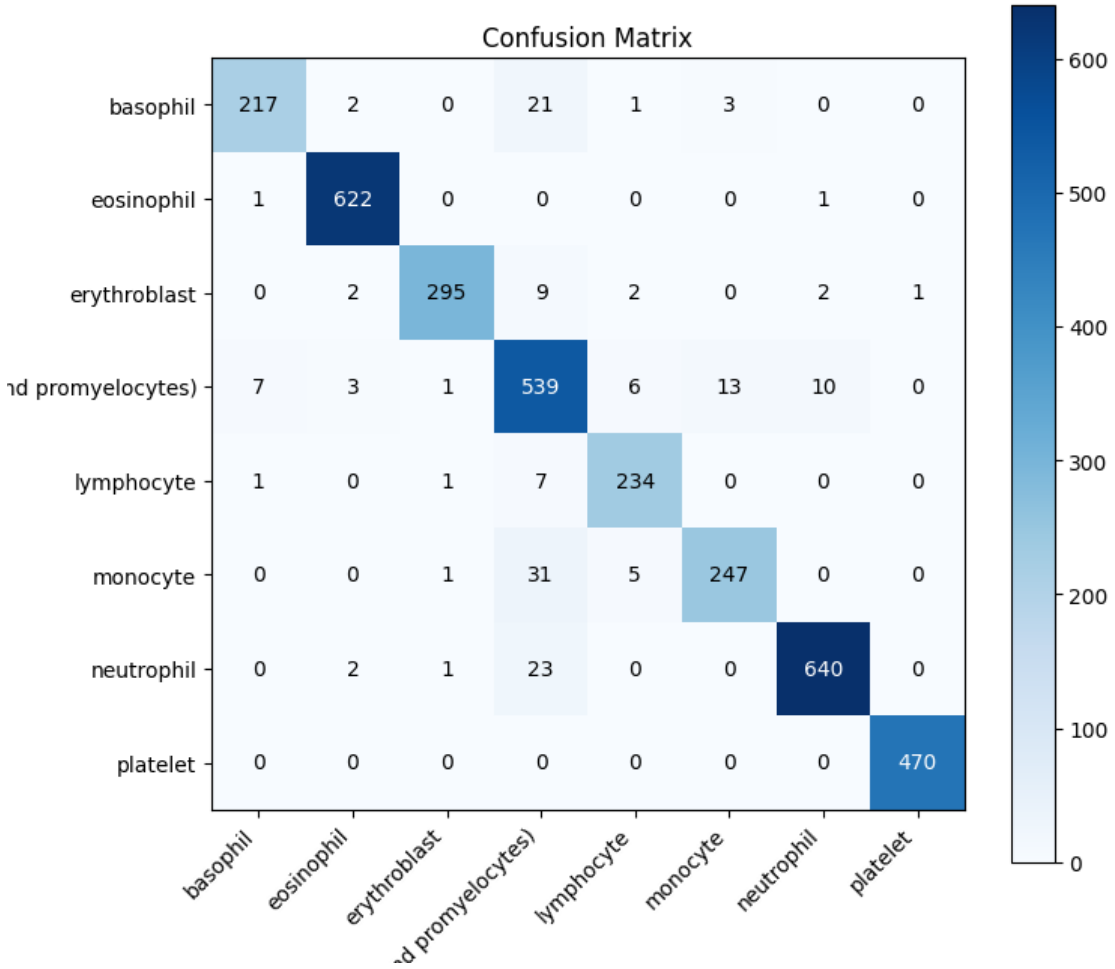
Μοντέλο	BN	Dr	WD(λ)	lr	ep	Tr Acc	Val Acc	Val Loss
1	×	×	×	1e-3	23	0.96	0.935	0.2
2	✓	×	×	5e-4	5	0.95	0.93	0.195
3	LN	×	×	5e-4	9	0.965	0.93	0.189
4	×	0.2	×	1e-3	25	0.936	0.93	0.186
4	×	0.5	×	1e-3	28	0.938	0.943	0.167
4	×	0.7	×	1e-3	28	0.91	0.935	0.2
5	×	×	1e-4	1e-3	18	0.934	0.93	0.208
6	✓	0.5	×	1e-4	29	0.966	0.947	0.149
7	✓	0.5	1e-4	1e-3	13	0.108	0.09	2.237
8	✓	0.5	×	1e-4	46	0.98	0.954	0.139

Best Model:

Μοντέλο	BN	Dr	lr	ep	Train Acc	Train Loss	Val acc	Val Loss	Test Acc
9	✓	0.5	1e-4	49	0.9512	0.1417	0.9574	0.1264	**0.954**







3.1 Παρατηρήσεις

- Το dataset ήταν πολύ εύκολο για το μοντέλο, αφού οι αποδόσεις του ήταν πολύ καλές(πάνω από 90%) από το πρώτο κι όλας μοντέλο που περιλάμβανε μόνο τη βασική αρχιτεκτονική και καμία τεχνική βελτίωσης. Οι διαφορές στο μοντέλο με κάθε τεχνική ήταν μικρές. Όπω είδαμε, το πρώτο και θεωρητικά χειρότερο μοντέλο συγκριτικά με το τελευταίο και θεωρητικά καλύτερο, έχουν μόνο 2% διαφορά στο validation accuracy, ενώ 6% στο validation loss.
- Το batch Normalization αύξησε την εκπαίδευση σχεδόν στο 100%, απλά το Early Stopping έτυχε να το κόψει νωρίς, γι'αυτό δε φαίνεται στην αναφορά.
- Το dropout πράγματι βοηθάει στη γενίκευση, ωστόσο όσο πιο μεγάλο το dropout rate, τόσο πέφτει η εκπαίδευση (διότι απενεργοποιούνται πιο πολλοί νευρώνες).
- Το Layer Normalization είχε παρόμοια συμπεριφορά με το Batch Normalization.

- Το μοντέλο 7, με το dropout και το weight Decay, μας δείχνει ότι το reguralization ήταν υπερβολικό και το μοντέλο κατέρρευσε.
- Το Data Augmkentation βοήθησε τη γενίκευση ελάχιστα (val loss έπεσε κατά 1%, και val acc ανέβηκε ελάχιστα), ωστόσο έκανε την εκπαίδευση πιο δύσκολη και γι'αυτό έπεσε σε ακρίβεια. Επίσης χρειάστηκε παραπάνω χρόνο στην εκπαίδευση.
- Τα datasets ήταν πολύ περίεργα. Όπως θα δείτε στο notebook, δεν έχω κάνει hard code για το συγκεκριμένο Dataset, για να μπορώ να δοκιμάζω και τα υπόλοιπα (άλλαζα μόνο το dataset name). Δοκίμασα αρκετά (PathMnist, dermamnist, octmnist, organmnist). Αρχικά είχα κάνει την εργασία πάνω στο dermamnist, όμως στο τέλος κοιτώντας τον confusion matrix, είχε σε ένα στοιχείο της διαγωνίου 1200 και στα άλλα πολύ μικροτερες τιμές. Και είδα μετά ότι το dataset περιέιχε και στο train, και στο validation, και στο test set πολύ περισσότερες εικόνες στη μία κλάση απ'οτι στις άλλες. Έτσι ότι απόδοση έβλεπα πριν στα μοντέλα, βασιζόταν κυρίως στο πόσο καλά έκανε προβλέψεις για αυτή τη κλάση. Άλλα όπως το pathmnist και το octmnist ήταν πολύ μεγάλα και οι εκπαιδεύσεις έπαιρναν ώρα. Το pathmnist το δοκίμασα για μερικά μοντέλα αλλά είχε απο την αρχή αποδόσεις τύπου 95-98% ακρίβεια γενίκευσης, οπότε δεν θα καταλαβαίναμε τις επιρροές των τεχνικών.
- Δυστυχώς δεν είχα χρόνο για περισσότερο πειραματισμό (batch sizes, learning rate, optimizer, epochs κ.λπ). Ωστόσο είχα πειραματιστεί με τις βασικές υπερπαραμέτρους και σε άλλο μάθημα οπότε δεν ήθελα να τα ξανακάνω.
- Ο confusion Matrix είναι ένας πίνακας, με γραμμές τις πραγματικές ταμπέλες των εικόνων, και στήλες τις προβλέψεις του μοντέλου. Για παράδειγμα, ενα στοιχείο που βρίσκεται στη γραμμή cat και στήλη dog, δείχνει πόσες εικόνες ήταν γάτες αλλά το μοντέλο μάντεψε σκύλους. Τα στοιχεία στις διαγωνίους είναι οι σωστές προβλέψεις του μοντέλου. Ο πίνακας αναφέρεται στις προβλέψεις πάνμω στο test set.

4 Part 2 - Transfer Learning on ResNet 18

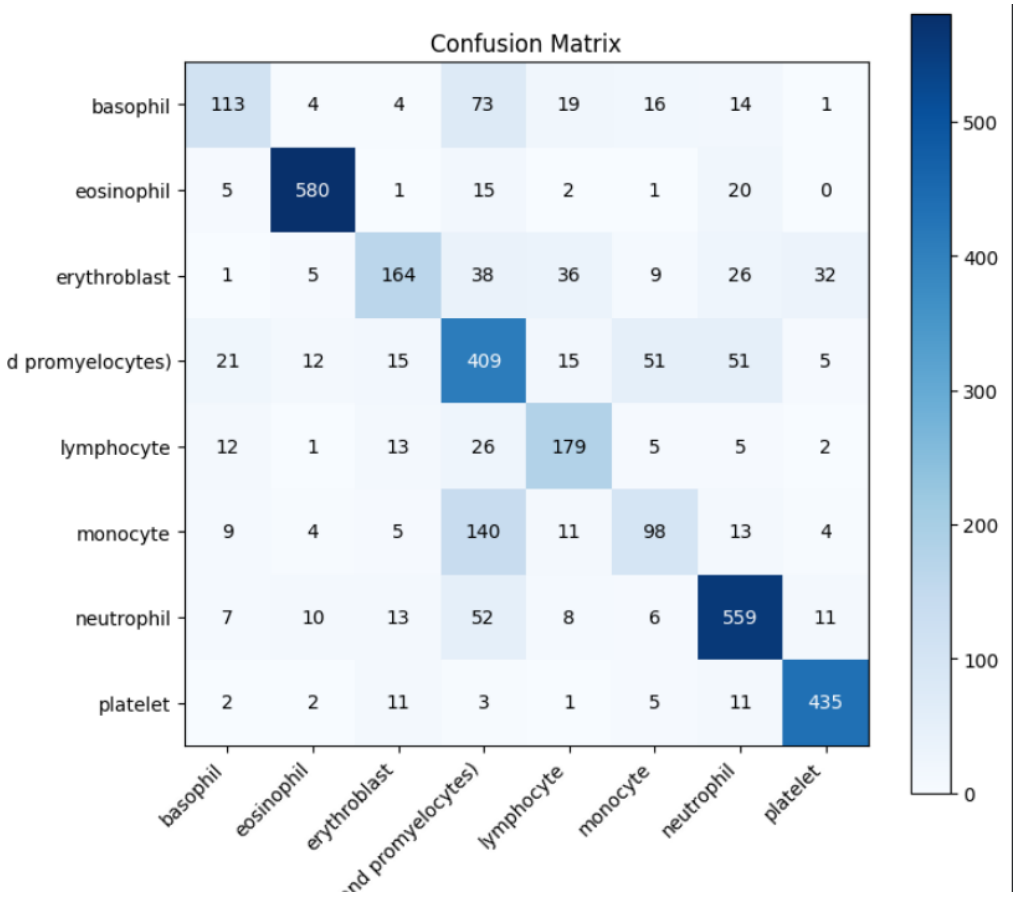
Το concept του transfer learning είναι να πάρεις ένα έτοιμο εκπαιδευμένο μοντέλο, να το προσαρμόσεις στο dataset σου και να δεις πως αποδίδει. Το προεκπαιδευμένο μοντέλο που διάλεξα ήταν το ResNet18, το οποίο έχει εκπαιδευτεί πάνω στο ImageNet με περίπου 1.2 εκατομμύρια εικόνες Σε αυτό το δοκίμασα έκανα 4 πράγματα:

- Πήρα τον classifier head (που είναι ο κατηγοριοποιητής, δηλαδή το τελευταίο γραμμικό (Dense/Linear) Layer που ουσιαστικά κάνει την κατηγοριοποίηση (δίνει τα logits), έβαλα 7 νευρώνες (γιατί έχω 7 κλάσεις) και τον εκπαίδευσα από την αρχή ξεπαγώνοντας τα βάρη του (δηλαδή τα έκανα εκπαιδεύσιμα). Τα υπόλοιπα βάρη ήταν παγωμένα.
- Έκανα fine tuning, δηλαδή πέρα απο το classifier head, ξεπάγωσα περισσότερα Layers του μοντέλου και τα εκπαίδευσα κι αυτά.
  - Εκπαίδευσα head + layer 4 (last)
  - Classifier head + layer 4 + layer 3
  - Classifier head + layer 4 + layer 3 + layer 2

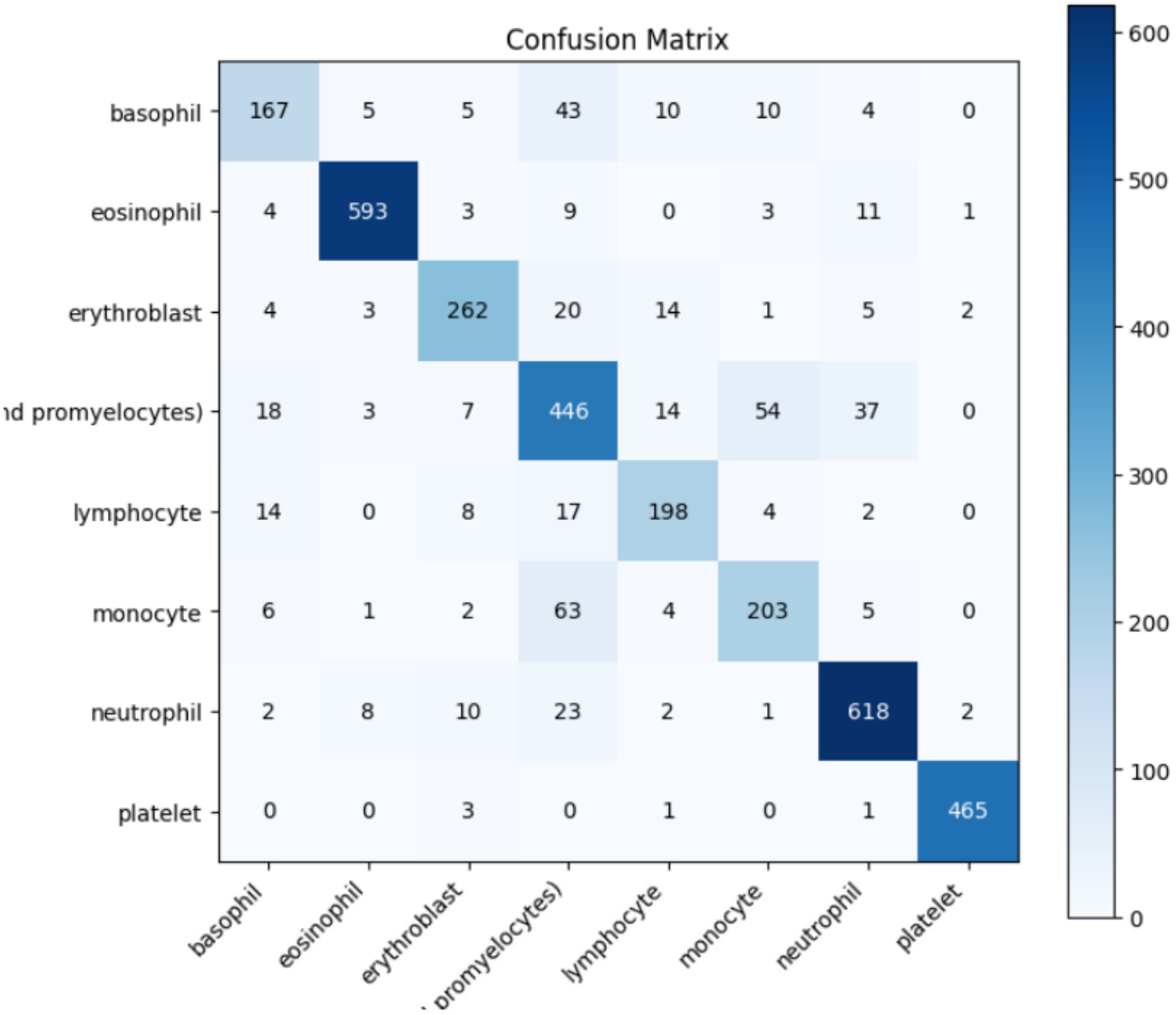
  Όπου γράφω layer προφανώς εννοώ το block layer του μοντέλου, και όχι ένα μεμονωμένο επίπεδο.

4.1 Αποτελέσματα

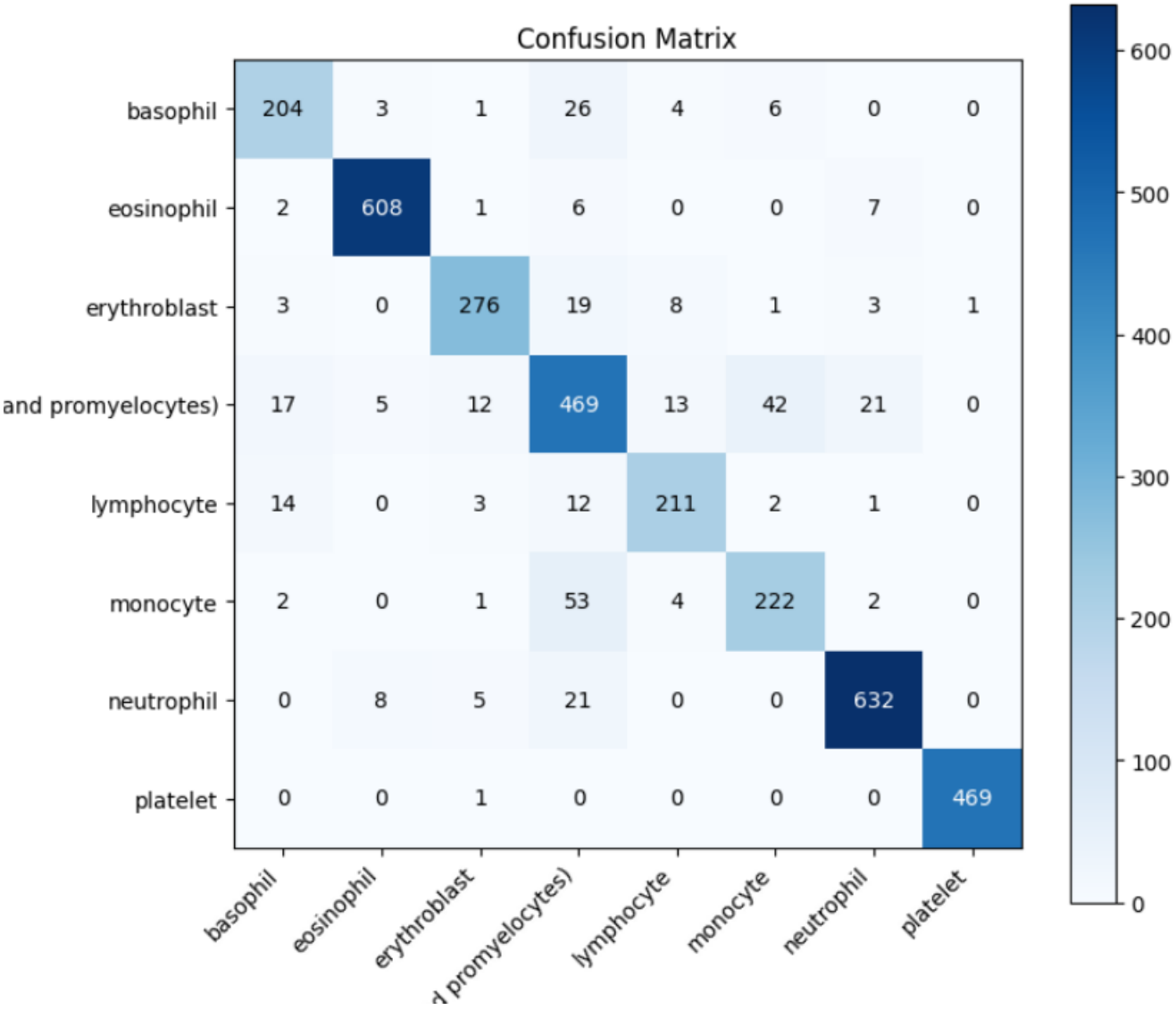
ResNet Training	Training Accuracy	Validation Accuracy	Validation Loss
Classifier head only	0.7436	0.76	0.699



ResNet Training	Training Accuracy	Validation Accuracy	Validation Loss
Classifier head + layer 4	0.91	0.8645	0.3769

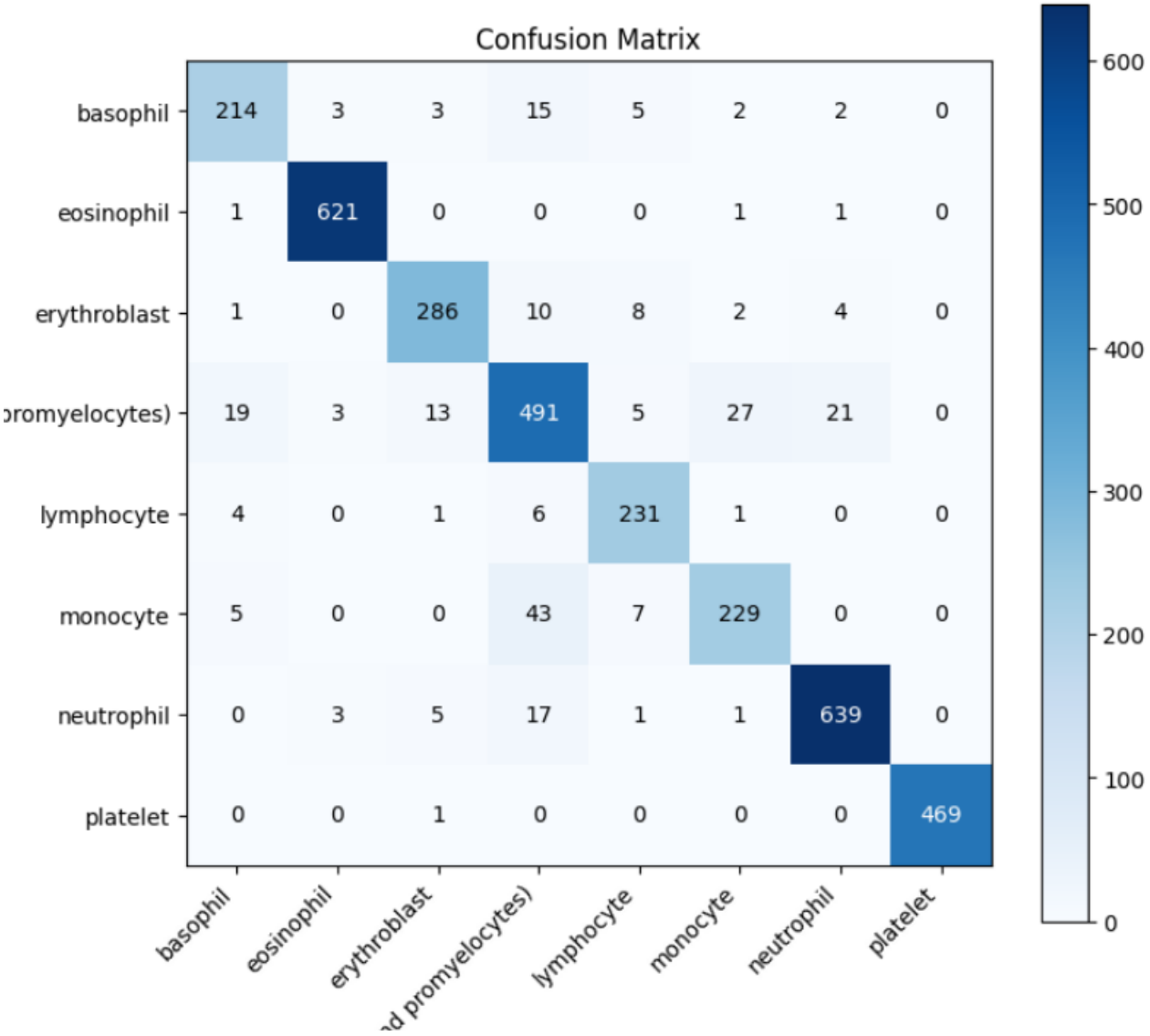


ResNet Training	Training Acc	Validation Acc	Validation Loss
Classifier head + layers 3,4	0.9444	0.9124	0.2553



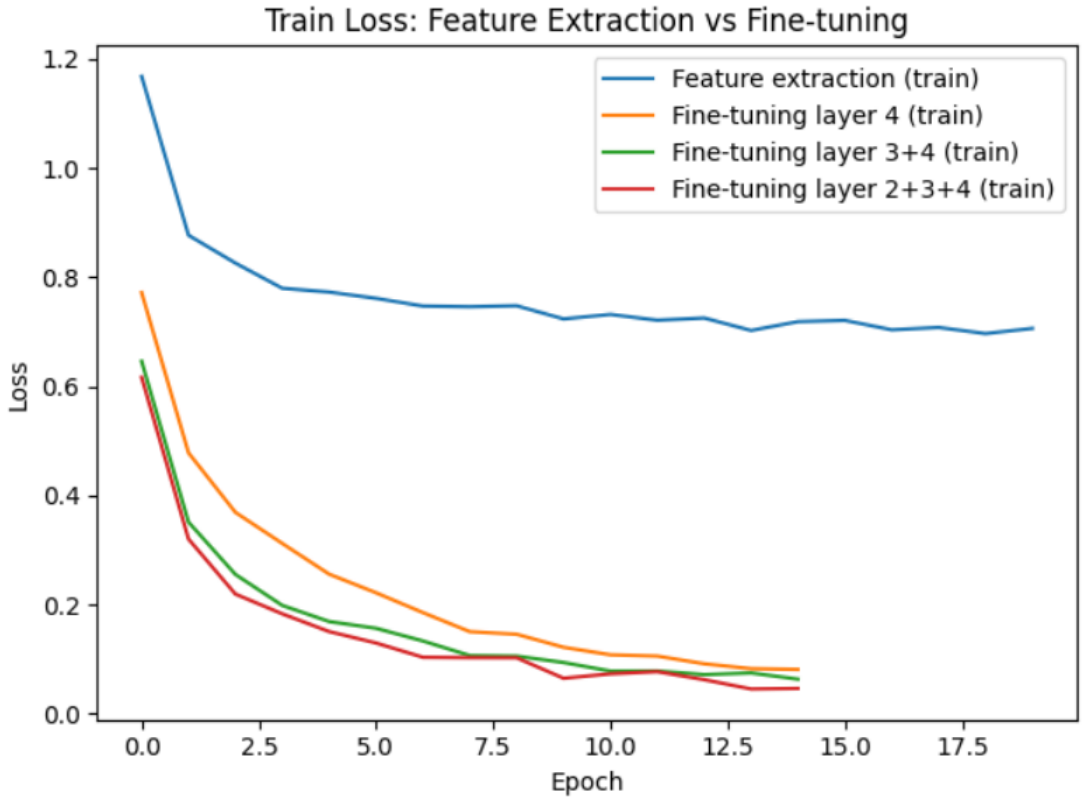
Best Model:

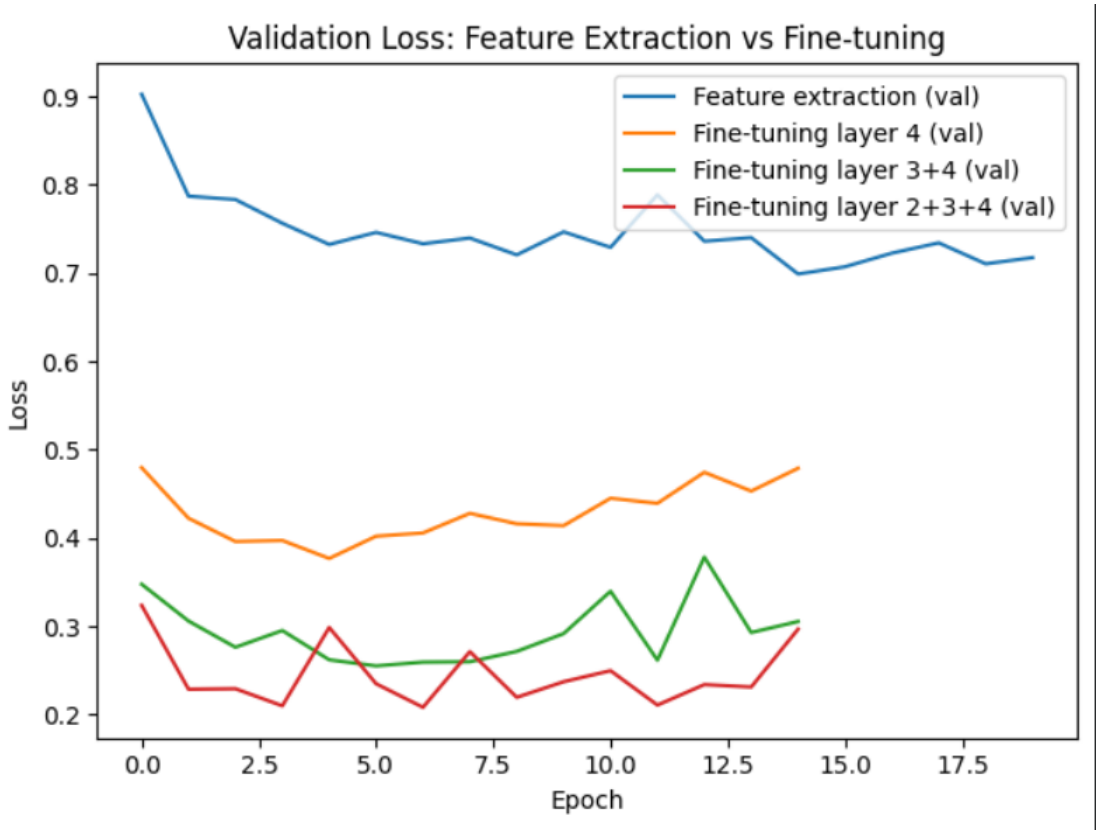
ResNet Training	Training Acc	Validation Acc	Val Loss	Test Acc
Classifier head + layers 2,3,4	0.9653	0.9387	0.2083	**0.9295**



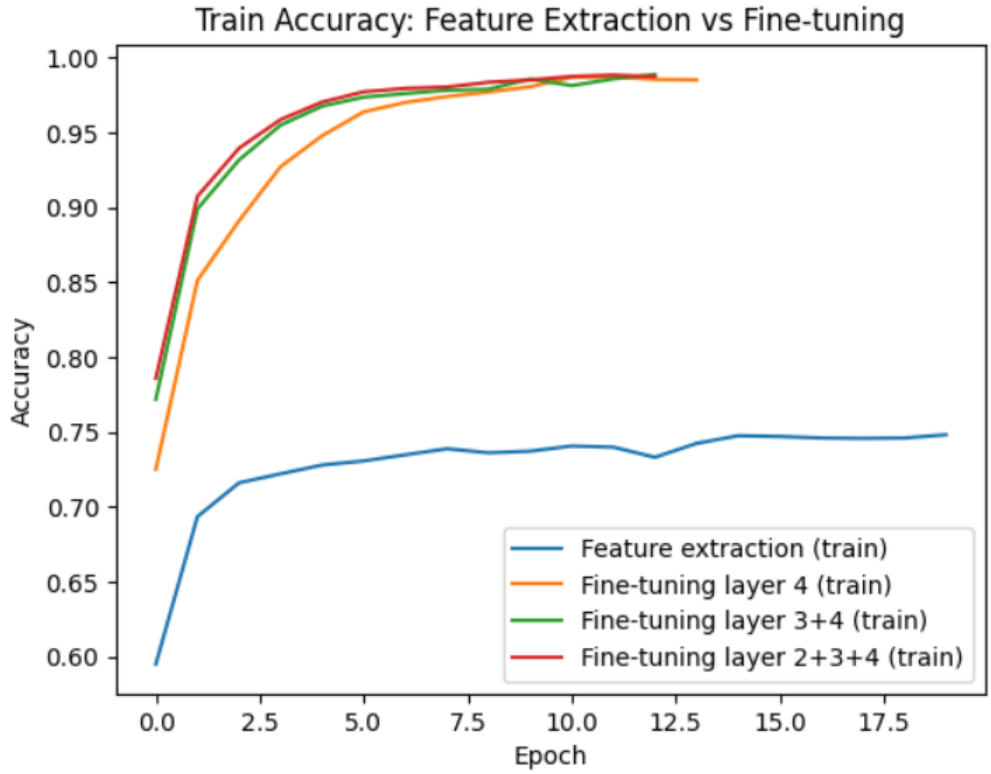
Learning Curves με όλες τις περιπτώσεις για σύγκριση:

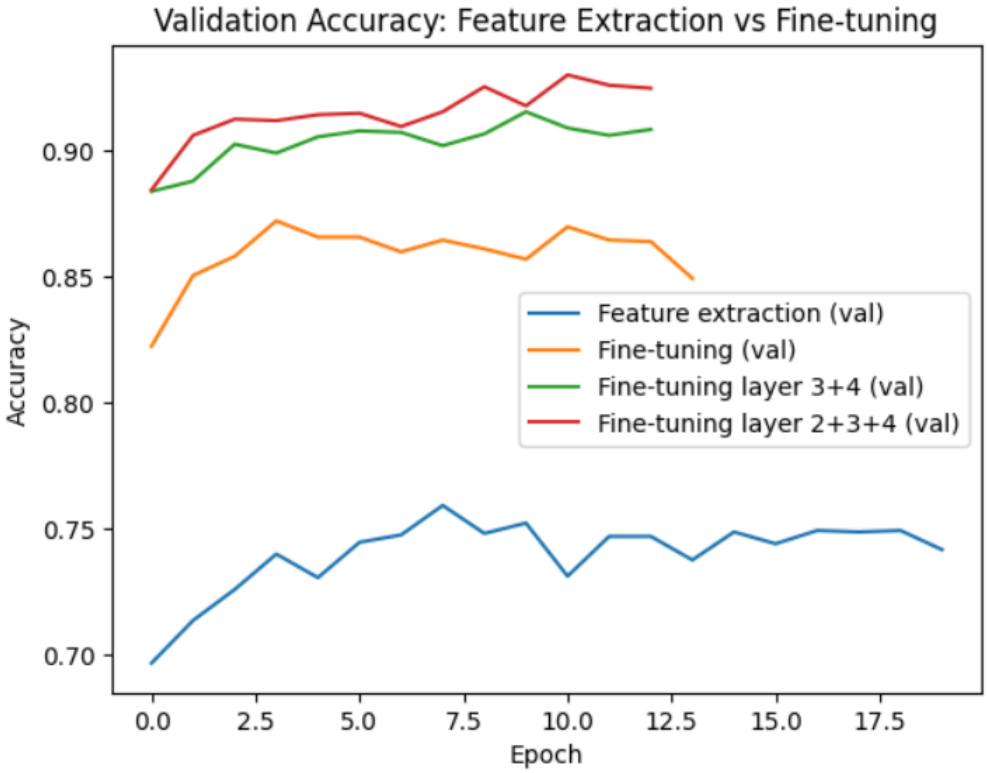
Loss Curves





Accuracy Curves





Παρατηρούμε πως το fine-tuning βελτίωσε πολύ το μοντέλο. Μάλιστα για κάθε layer που ξεπάγωνα, υπήρχε βελτίωση, ειδικά στο πρώτο και layer 4, και

στο 3+4. Ωστόσο η ταχύτητα των εκπαιδεύσεων ανέβηκε πολύ, προφανώς λόγω της πολυπλοκότητας του μοντέλου.

Να σημειώσω οτι το μοντέλο δεν έχει εκπαιδευτεί σε ιατρικές εικόνες, και πάλι έδωσε απόδοση 75% αρχικά. Προφανώς, με την εκπαίδευση ενός και δύο blocks του, το μοντέλο προσαρμόστηκε στο dataset, έμαθε, και έδωσε πολύ καλύτερα αποτελέσματα.

Θα ήθελα να δω τι θα γινόταν αν ξεπάγωνα και το 1ο layer, αλλά λόγω χρόνου δεν το έκανα.

5 Transfer Learning on Transformer DeIT

Ίδια φιλοσοφία με το Part 2 και εδώ, απλά αντί CNN, θα χρησιμοποιήσουμε έναν προεκπαιδευμένο Transformer, και συγκεκριμένα ένα μοντέλο DeIT (Data efficient Image Transformer). Οι DeITs είναι βελτιωμένες εκδοχές των VIT σε μικρά datasets. Ωστόσο, εδώ χρειάστηκε να κάνουμε resize τις αρχικές εικόνες σε 224x224, για να ταιριάζουν με το μέγεθος που έχει εκπαιδευτεί ο Transformer.

5.1 Vision Transformers (ViT)

Οι Vision Transformers (ViT) είναι Transformers φτιαγμένοι για κατηγοριοποίηση εικόνων. Δε χρησιμοποιούν συνελίξεις όπως τα CNNs. Η καινοτόμα ιδέα πάνω τους ήταν το self-attention.

5.1.1 Patch Embeddings

Έστω μια έγχρωμη εικόνα διαστάσεων $H \times W \times C$ (π.χ. $224 \times 224 \times 3$). Η εικόνα χωρίζεται σε μη επικαλυπτόμενα patches σταθερού μεγέθους $P \times P$. Ο συνολικός αριθμός patches δίνεται από:

$$N = \frac{H}{P} \cdot \frac{W}{P}$$

Για παράδειγμα, για $P = 16$, έχουμε $14 \times 14 = 196$ patches.

Κάθε patch $x_i \in \mathbb{R}^{P \times P \times C}$ γίνεται flatten και μετατρέπεται σε ένα διάνυσμα:

$$x_i \in \mathbb{R}^{P^2 \cdot C}$$

Στη συνέχεια, εφαρμόζεται ένας γραμμικός μετασχηματισμός (linear projection):

$$z_i = x_i W_E + b_E, \quad z_i \in \mathbb{R}^D$$

Τα z_i λέγονται patch embeddings και λειτουργούν ως tokens εισόδου στον Transformer.

5.1.2 Positional Embeddings

Επειδή οι Transformers δεν καταλαβαίνουν τη χωρική διάταξη των δεδομένων, χρησιμοποιούν positional embeddings ώστε να κωδικοποιούν τη θέση κάθε patch στην εικόνα. Για κάθε patch embedding z_i προστίθεται ένα διάνυσμα θέσης p_i:

$$\tilde{z}_i = z_i + p_i$$

Το διάνυσμα θέσης p_i είναι το positional embedding. Τα positional embeddings μαθαίνονται στην εκπαίδευση και επιτρέπουν στο μοντέλο να συσχετίζει χωρικά τα patches.

5.1.3 Self-Attention

Ο βασικός μηχανισμός επεξεργασίας στα ViT είναι η self-attention. Για την ακολουθία των positional embeddings $Z = [\tilde{z}_1, \ldots, \tilde{z}_N]$, υπολογίζονται τρεις διαφορετικές αναπαραστάσεις:

$$Q = ZW_Q, \quad K = ZW_K, \quad V = ZW_V$$

όπου W_Q, W_K, W_V είναι πίνακες βαρών. Η self-attention ορίζεται ως:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

και συσχετίζει καθε $\hat{z}_i$ με όλα τα άλλα. Έτσι το μοντέλο καταλαβαίνει καλύτερα πως να ερμηνεύσει αυτό που πρέπει (έχει context).

Συνήθως οι VIT απαιτούν μεγάλα σύνολα δεδομένων για να μάθουν αποτελεσματικά και τείνουν να υπερπροσαρμόζονται όταν εκπαιδεύονται από το μηδέν σε μικρά datasets.

5.2 Data-efficient Image Transformer (DeiT)

Το DeiT (Data-efficient Image Transformer) έρχεται ως λύση στο πρόβλημα που αντιθμετωπίζουν οι VIT στα μικρά datasets.

Χρησιμοποιεί τεχνικές knowledge distillation, όπου ένας ισχυρός teacher model (συνήθως CNN) καθοδηγεί την εκπαίδευση του Transformer. Με αυτόν τον τρόπο, το μοντέλο μαθαίνει όχι μόνο από τα πραγματικά labels, αλλά και από τις προβλέψεις του teacher, γεγονός που βελτιώνει σημαντικά τη γενίκευση.

Έτσι, τα DeiT μοντέλα μπορούν να εκπαιδευτούν αποδοτικά με λιγότερα δεδομένα και παρουσιάζουν καλύτερη απόδοση σε μικρά datasets.

6 Αποτελέσματα

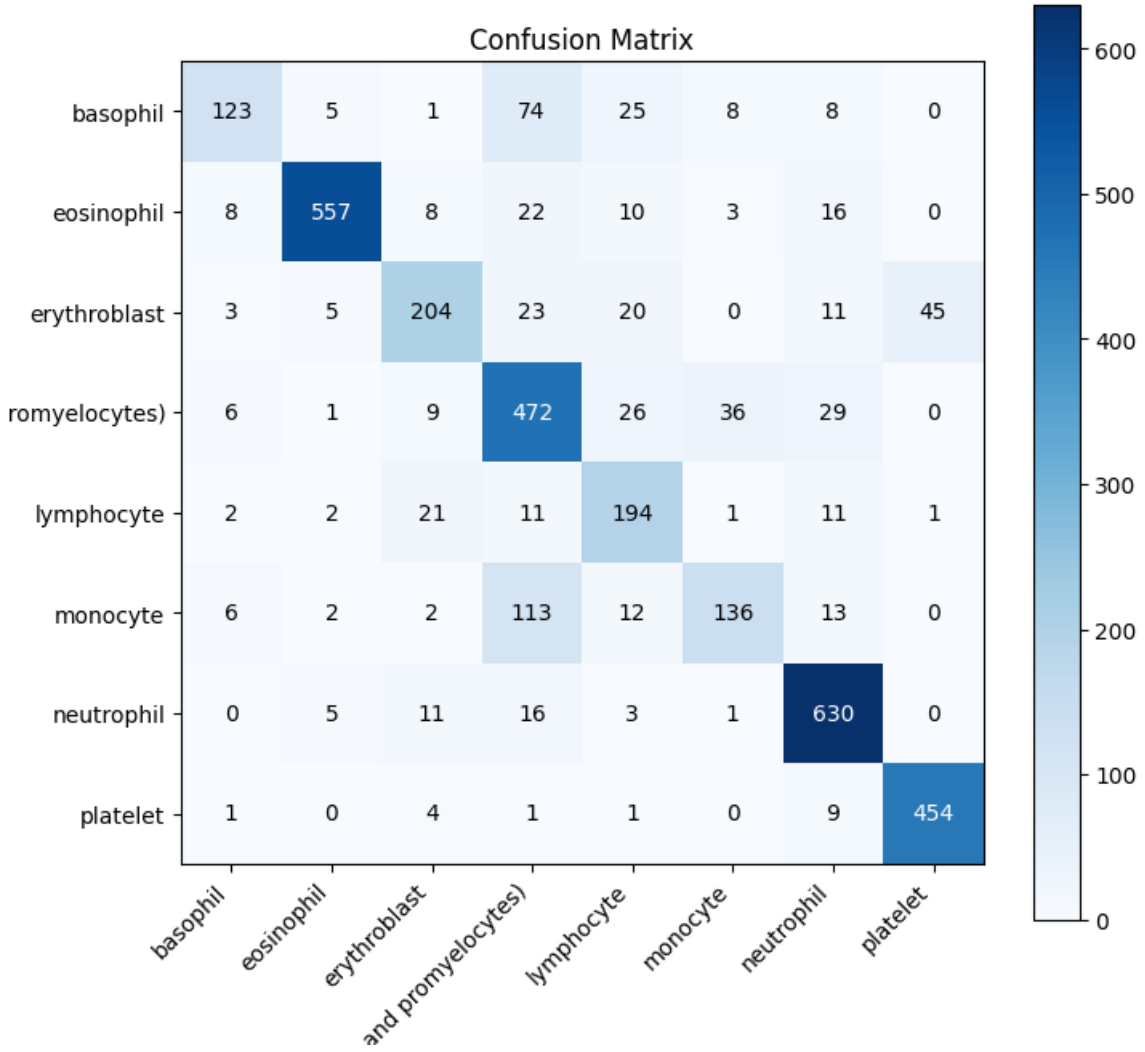
Κι εδώ έκανα 2 μέρη:

- Εκπαίδευση μόνο του classifier head για κατηγοριοποίηση σε 7 κλάσεις
- Fine-tuning εκπαιδεύοντας τα 2 τελευταία layer blocks μαζί με τον classifier head

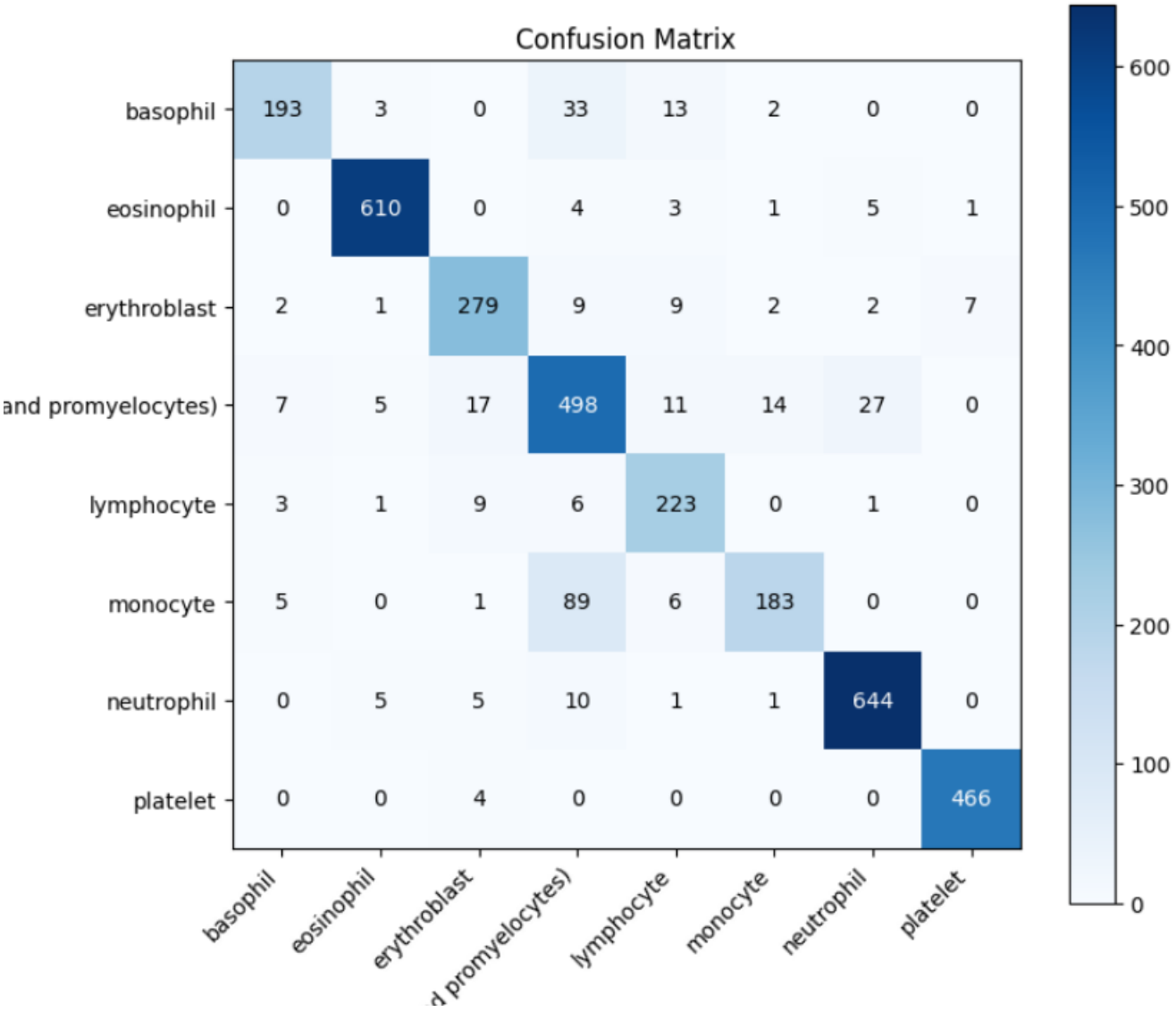
Σημειώνω ότι μειώσαμε το batch size (από 64 σε 32) και ειδικά στο fine tuning (16). Αυτό έγινε σκόπιμα, λόγω του οτι δουλεύουμε σε 224x224 εικόνες και επειδή οι transformers απαιτούν πολλή μνήμη.

Συγκεκριμένα όμω σγια το fine-tuning, ρίχνουμε το batch size και για άλλον ενα λόγο. Όσο πιο μικρό το batch size, τόσο πιο θορυβώδη είναι το gradient του κάθε batch (επειδή το gradient του συνολικού loss για το κάθε batch θα είναι μέσος όρος λιγότερων δειγμάτων). Ο θόρυβος αυτός βοηθάει στη γενίκευση του μοντέλου, γιατί το βοηθάει να ξεφύγει από sharp minima (δηλαδή τοπικά ελάχιστα στα οποία φτάνεις με απότομη κλίση. Αυτά τα minima είναι κακά γιατί με την παραμικρή αλλαγή των δεδομένων, το loss αυξάνεται απότομα. Δηλαδή το μοντέλο έχει υπερπροσαρμοστεί στα δεδομένα εκπαίδευσης). Στο fine tuning, επειδή ξεπαγώνουμε πολλά βάρη, ο χώρος των μεταβλητών γίνεται υψηλής διάστασης και είναι πολύ πιο εύκολο να πέσουμε σε sharp minima. Με μικρότερα batches, κάνουμε πιο πολλές ενημερώσεις στα βάρη, η αναζήτηση είναι πιο στοχαστική και γίνεται καλύτερη εξερεύνηση του πεδίου λύσεων (w).

DeiT Training	Training Acc	Validation Acc	Validation Loss
Classifier head	0.7154	0.7926	0.5807

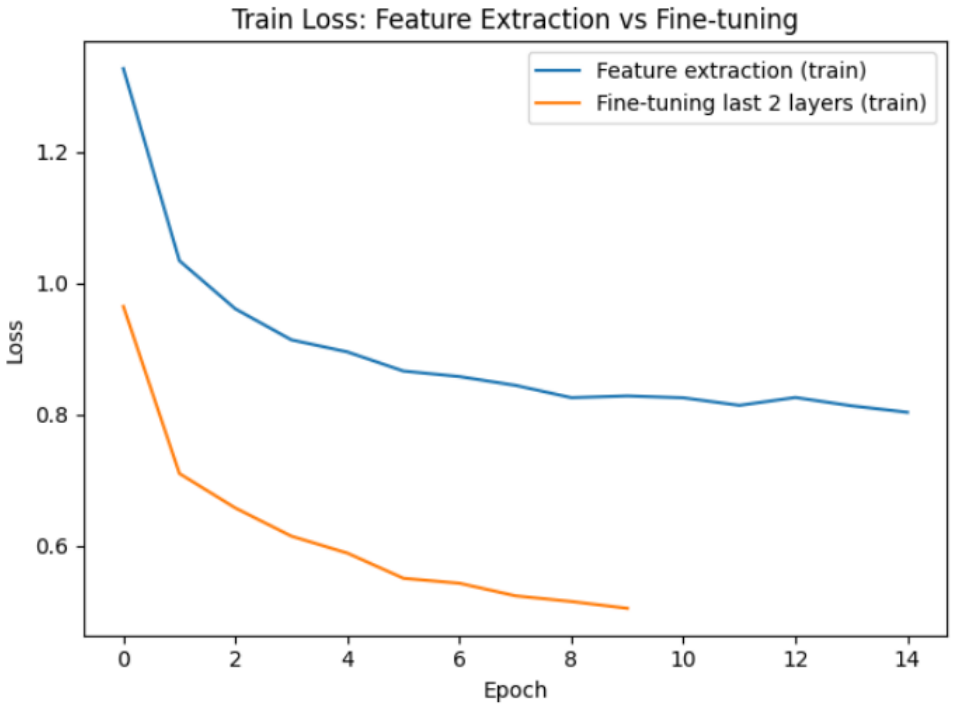


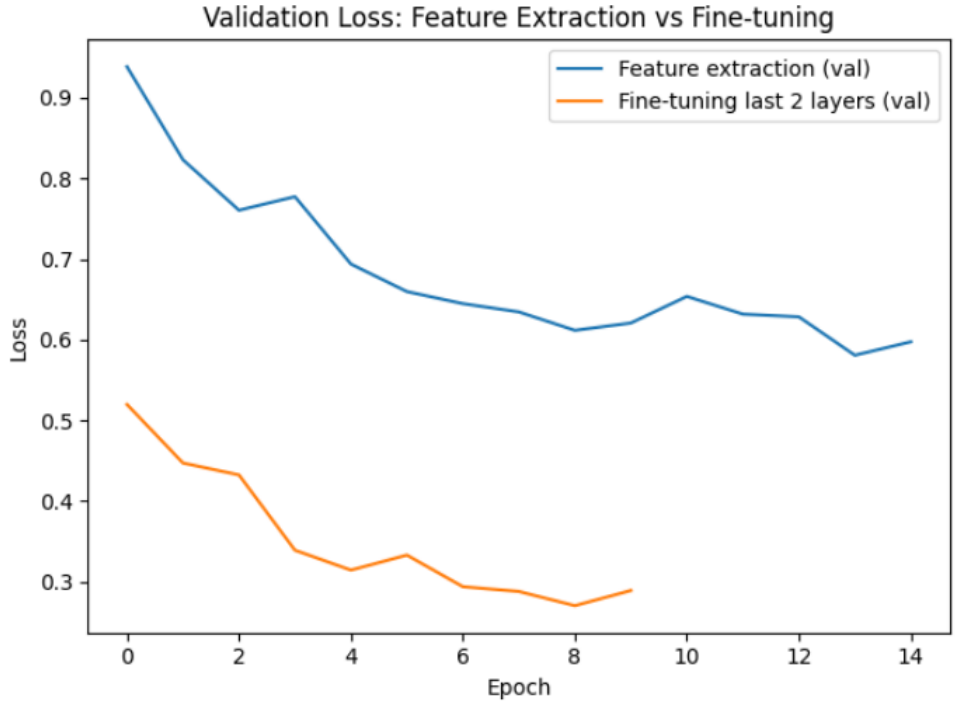
DeiT Training	Training Acc	Validation Acc	Val Loss	Test Acc
Classifier head + last 2 layers	0.8129	0.9019	0.2702	**0.905**



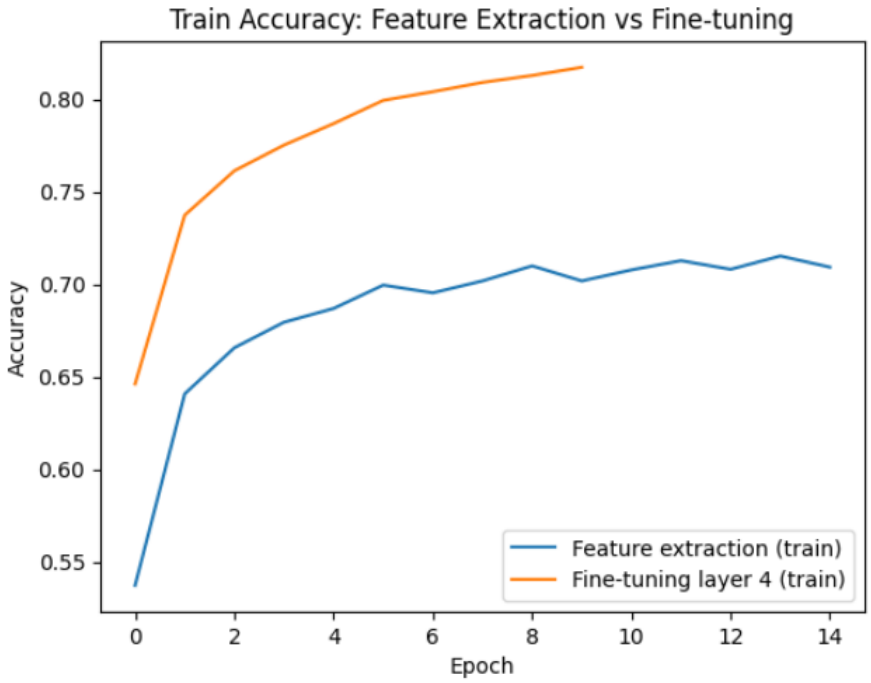
Learning Curves:

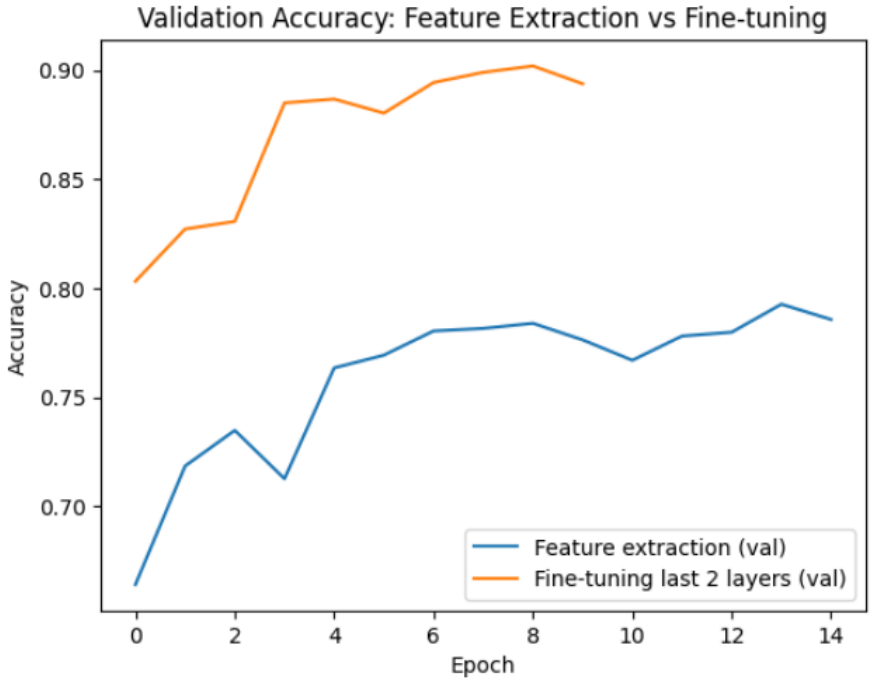
Loss Curves





Accuracy Curves





6.1 Παρατηρήσεις

- Βλέπουμε πάλι την μεγάλη βελτίωση που έφερε το fine tuning
- Μου έκανε εντύπωση που το gap μεταξύ training-validation accuracy είναι αρκετά μεγάλο στον transformer (8-9%) με την εκπαίδευση να υστερεί. Μάλλον οφείλεται στο ότι ο transformer περιέχει Data Augmentations και Dropouts κ.λπ.
- Οι χρόνοι εκπαίδευσης ήταν οι μεγαλύτεροι με διαφορά, σε σχέση με όλα τα υπόλοιπα μοντέλα, ειδικά στο fine tuning, και γι'αυτό το έτρεξα μόνο για 10 εποχές.