

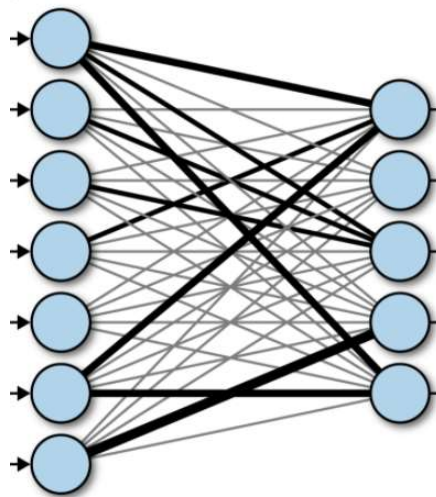
Όραση Υπολογιστών: Τεχνική Αναφορά 4^{ης} Εργασίας

Παπαδόπουλος Αριστείδης Α.Μ.:57576

1. Μη Προ-εκπαιδευμένο δίκτυο:

- **Είδη layers που χρησιμοποιήθηκαν:**

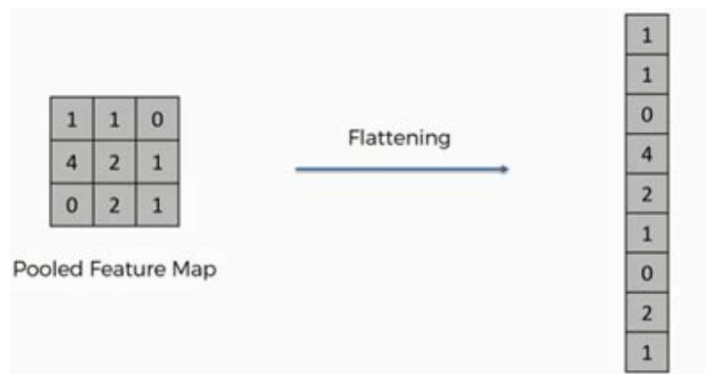
- Dense Layer(tf.keras.layers.Dense(units, activation...)):



Εικ.1 Παράδειγμα Dense (Fully Connected) Layer.

Το πιο απλό είδος layer,κάθε νευρώνας του layer συνδέεται με όλους τους προηγούμενους νευρώνες του προηγούμενου layer, επομένως για N νευρώνες, προκύπτουν 2^N συνδέσεις. Λόγω των συνδέσεων αυτών, εμφανίζονται πάρα πολλές συνδέσεις, για τις οποίες χρειάζεται υπολογιστική ισχύς για να υπολογίζονται και να ανανεώνονται τα βάρη της κάθε σύνδεσης. Η παράμετρος `units` είναι το πλήθος των νευρώνων που θα έχει το layer, ενώ η παράμετρος `activation` αναφέρεται στην συνάρτηση ενεργοποίησης που χρησιμοποιείται. Οι συναρτήσεις αυτές χρησιμοποιούνται ως υπερπαράμετροι και στα υπόλοιπα layers, κυρίως στα συνελκτικά. Υπάρχουν και άλλοι υπερπαράμετροι, οι οποίες όμως δεν χρησιμοποιούνται και παίρνουν τις default τιμές τους.

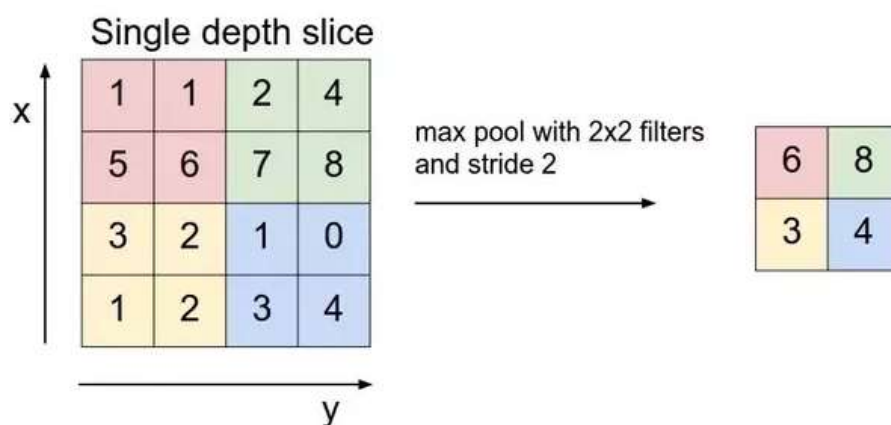
➤ Flatten Layer(`tf.keras.layers.Flatten()`):



Εικ.2 Παράδειγμα που αναπαριστά την διαδικασία που υλοποιεί το Flatten Layer.

Μετατρέπει την είσοδο του (μια εικόνα) σε ένα διάνυσμα ίσο με το γινόμενο των διαστάσεων της εικόνας. Για παράδειγμα μια εικόνα (40,40,2) , στην έξοδο θα μετατραπεί σε διάνυσμα με μέγεθος 3200. Χρησιμοποιείται χωρίς κάποια υπερπαράμετρο, παίρνει default τιμή η μοναδική του υπερπαράμετρος.

➤ Max-Pooling Layer (`tf.keras.layers.MaxPooling2D(pool_size, strides, padding)`):

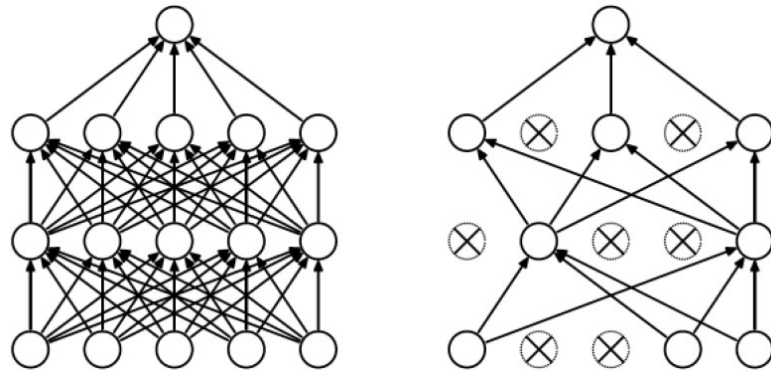


Εικ.3 Παράδειγμα που αναπαριστά την διαδικασία που υλοποιεί το Max-Pooling Layer.

Μειώνει τις διαστάσεις της εισόδου του, καθώς αυτή διατρέχεται μέσα στο νευρωνικό δίκτυο. Η μείωση ουσιαστικά γίνεται με υποδειγματοληψία και με αυτόν τον τρόπο, μειώνεται η υπολογιστική δύναμη που χρειάζεται κατά την διάρκεια της εκπαίδευσης. Το ποσοστό μείωσης εξαρτάται από τις διαστάσεις του φίλτρου που ορίζεται και από το και το πόσο μετακυλήσει το παράθυρο(`strides`) που χρησιμοποιείται σε κάθε διάσταση για να κάνει την επόμενη υποδειγματοληψία. Χρησιμοποιούνται μετά τα convolutional layers. Η δειγματοληψία ανάλογα με το είδος, επιλέγει την μέγιστη τιμή ή την μέση από αυτές που βλέπει σε κάθε

πέρασμα. Εγώ επέλεξα να χρησιμοποιήσω μόνο το `max_pool` σε όλες τις αρχιτεκτονικές και σε όλες τις δοκιμές.

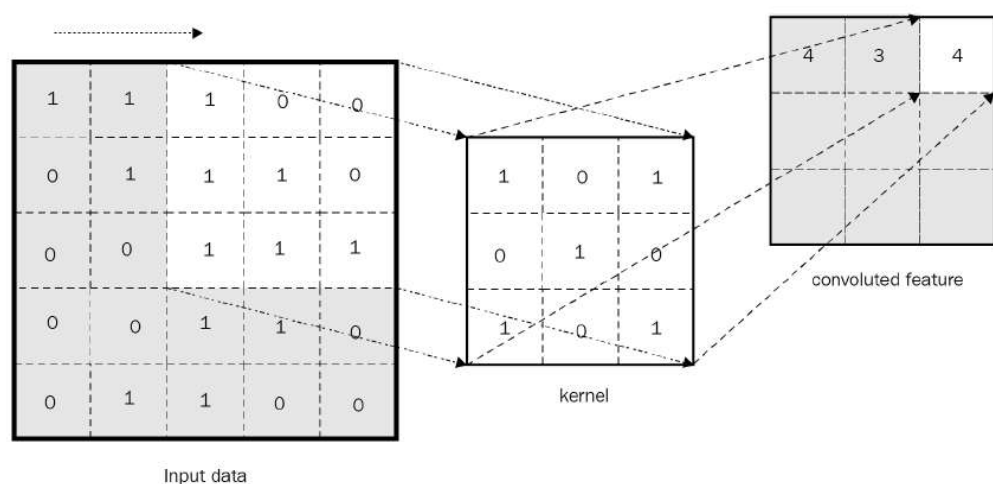
➤ `Drop-Out Layer(tf.keras.layers.Dropout(rate))`:



Εικ.4 Παράδειγμα που αναπαριστά την διαδικασία που υλοποιεί το Drop-Out Layer.

Ελαχιστοποιεί το φαινόμενο over-fitting, αφαιρώντας τον πιθανό θόρυβο που βρίσκεται στις εικόνες. Αυτό γίνεται ορίζοντας τυχαία κάποια layer ίσα με 0, απορρίπτοντας καθ' αυτόν τον τα αντίστοιχα layer με τυχαίο τρόπο. Έτσι, απλοποιείται η αρχιτεκτονική και αποφεύγεται το over-fitting. Το `rate` παίρνει τιμές μεταξύ 0-1 και καθορίζει τον ρυθμό με τον οποίο θα ορίζονται τυχαία τα layers ως 0. Όσο μεγαλύτερο το `rate`, τόσο περισσότεροι νευρώνες απορρίπτονται.

➤ `Convolutional(tf.keras.layers.Conv2D(filters, kernel_size, strides, padding, activation...))`:



Εικ.5 Παράδειγμα που αναπαριστά την διαδικασία που υλοποιεί το Convolutional Layer(για πυρήνα 3x3).

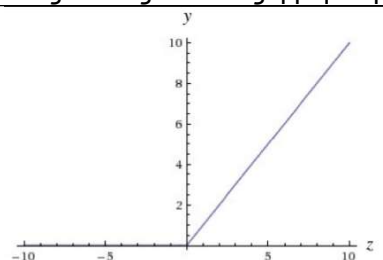
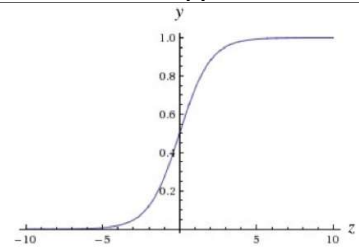
Υλοποιεί συνέλιξη στην είσοδο του, με αποτέλεσμα να μειώνονται οι διαστάσεις της εισόδου. Προσπαθεί να συσχετίσει την πληροφορία που εντοπίζει σε τοπικό επίπεδο στην είσοδο τους, το οποίο στην συνέχεια δύναται να συσχετιστεί με κάποιο εκ των αποτελεσμάτων

των συνελίξεων. Η συνέλιξη γίνεται με πυρήνες και το αποτέλεσμα κάθε συνέλιξης καταχωρείται σε έναν χάρτη χαρακτηριστικών(feature map). Οι διαστάσεις του πυρήνα, οδηγούν σε αντίστοιχη μείωση των δύο πρώτων διαστάσεων, ενώ το πλήθος των πυρήνων που επιλέγονται, αυξάνουν την 3^η διάσταση(που αποτελεί την διάσταση των feature maps,δηλαδή το πλήθος τους), αφού γίνονται τόσες συνελίξεις όσοι και οι πυρήνες. Οι 2 πρώτες διαστάσεις μπορούν να παραμείνουν σταθερές, με την χρήση padding.Κάθε νευρώνας συνδέεται με το γινόμενο των διαστάσεων του πυρήνα, σε αντίθεση με το Dense layer.Έτσι, γίνεται έλεγχος σε μια μικρή περιοχή, όπου τα στοιχεία της περιοχής συνδέονται νοηματικά μεταξύ τους.

- **Υπερπαράμετροι για κάθε layer:**

- **Activation:**

Η συνάρτηση ενεργοποίησης χρησιμοποιείται προκειμένου να ορίζεται η έξοδος του νευρώνα για οποιαδήποτε είσοδο. Το είδος της συνάρτησης, επομένως επιδρά στην έξοδο καθώς προκύπτουν διαφορετικές τιμές για διαφορετικές συναρτήσεις. Μερικές συναρτήσεις είναι οι εξής:

Όνομα	Τύπος	Έξοδος - Επεξήγηση
Relu(Rectified Linear Unit)	$g(t) = \max(0, t)$	 <p>Ορίζει στις αρνητικές τιμές την τιμή 0,ενώ στις θετικές την τιμή που έχουν.</p>
Sigmoid	$g(t) = \frac{1}{1 + e^{-t}}$	 <p>Όταν η τιμή είναι πολύ μικρή, η έξοδος είναι κοντά στο 0,ενώ όταν είναι πολύ μεγάλη, κοντά στο 1.</p>

SoftMax	$g(f_1, \dots, f_c)$ $= \left(\frac{e^{f_1}}{\sum_j e^{f_j}}, \dots, \frac{e^{f_c}}{\sum_j e^{f_j}} \right)$	Μετατρέπει τις εισόδους σε πιθανότητες, προκειμένου να γίνει η κατηγοριοποίηση
---------	---	--

Επίσης, αξίζει να σημειωθεί πως τελικά παρόλο που υπάρχουν αρκετές συναρτήσεις ενεργοποίησης, έχει επικρατήσει σε σχεδόν καθολικό βαθμό η συνάρτηση Relu, την οποία και χρησιμοποιώ. Τέλος, βρήκα μετά από σχετική αναζήτηση τις συναρτήσεις "mish" και "leaky Relu", τις οποίες όμως δεν κατάφερα να τις παραμετροποιήσω προκειμένου να χρησιμοποιηθούν κατάλληλα.

➤ **Pool_size, stride, Padding, Filters, Kernel_size:**

Το pool_size καθορίζει το μέγεθος του παραθύρου στο οποίο θα γίνει η υποδειγματοληψία, ενώ η υπερπαραμέτρος strides καθορίζει το πόσο θα μετακυλήσει το παράθυρο μετά από κάθε υποδειγματοληψία. Αν χρησιμοποιηθεί padding, θα μεταβληθούν οι διαστάσεις της εξόδου προκειμένου να έχει ίδιες διαστάσεις με την είσοδο. Τα strides και τα padding χρησιμοποιούνται και στα Convolutional Layers, όπου τα strides δείχνουν πόσο θα μετακινηθεί ο πυρήνας για να κάνει την επόμενη συνέλιξη. Η υπερπαραμέτρος filters είναι ο αριθμός των φίλτρων/πυρήνων που χρησιμοποιούνται κατά την συνέλιξη, ενώ το kernel_size είναι το μέγεθος τους.

- **Γενικές υπερπαραμέτροι:**

➤ **Batch_size:**

Η τιμή του είναι πόσες εικόνες μαζί θα διαβάζονται από το αντίστοιχο directory, και στην συνέχεια θα περνάνε στο δίκτυο για εκπαίδευση. Εξαρτάται από το πόσες είναι οι διαθέσιμες εικόνες και από την μνήμη του συστήματος (πόσες μπορεί να διαβάσει ταυτόχρονα). Επίσης, επηρεάζει μαζί με το steps_per_epoch (βλ. παρακάτω) το πόσες εικόνες θα εισέρχονται ανά εποχή.

➤ **Shuffle, Class_mode, Rescale:**

Το shuffle είναι επιλογή που αν ενεργοποιηθεί, η ανάγνωση των εικόνων γίνεται με τυχαίο τρόπο και όχι με την σειρά, ενώ το class_mode επιλέγεται ως categorical, διότι έχουμε περισσότερες κλάσεις από 2. Τέλος, το rescale κανονικοποιεί την κάθε εικόνα, ώστε οι τιμές των πίξελ της να είναι μεταξύ 0 και 1.

➤ **Target_size:**

Καθορίζει τις διαστάσεις με τις οποίες θα διαβαστούν εικόνες. Πρέπει να επιλεγεί με τέτοιον τρόπο, ώστε να μπορούν να διακριθούν τα χαρακτηριστικά της κάθε εικόνας, καθώς μεγάλες τιμές κάνει stretch την εικόνα, ενώ μικρές τιμές την συμπιέζουν (ανάλογα με τις διαστάσεις της κάθε εικόνας). Όσο μεγαλύτερες οι διαστάσεις, τόσο περισσότερος χρόνος χρειάζεται για να διαβαστούν και να προσπελαστούν, καθυστερώντας συνολικά την εκπαίδευση, ενώ δεν είναι αναγκαίο να αυξηθεί και η συνολική απόδοση. Επέλεξα να είναι (100,100) ως μια μέση λύση μεταξύ των διαστάσεων των εικόνων των data set που είναι στην διάθεσή μου, αλλά και ως μέση λύση μεταξύ του χρόνου που χρειάζεται και της τελικής απόδοσης.

➤ **Validation_split & seed:**

Επειδή απουσιάζει ένα validation set, επέλεξα να διαχωρίσω το database της εκπαίδευσης, με ποσοστό 25%, δηλαδή επιλέχθηκαν τυχαία εικόνες από το TensorFlow/Keras για να τοποθετηθούν σε validation set, οι οποίες είναι το 25% των αρχικών εικόνων. Το seed χρησιμοποιείται με την τιμή 1, για να μην προσδώσω κάποιο βάρος στις εικόνες. Έτσι, αντί να εκπαιδεύεται το δίκτυο σε ένα μεγαλύτερο πλήθος εικόνων, διαχωρίζω ένα κομμάτι από αυτές, για να κάνω το validation και να ελεγχθεί πόσο καλή ή κακή ήταν η εκπαίδευση.

Τα παραπάνω αποτελούν υπερπαραμέτρους που καθορίζουν τον τρόπο που θα διαβαστούν οι εικόνες. Αντίθετα, οι παρακάτω υπερπαραμέτροι καθορίζουν τον τρόπο εκπαίδευσης του δικτύου.

➤ **Epoch:**

Πόσες φορές θα περάσουν οι εικόνες του training dataset μέσα από το δίκτυο. Δηλαδή, πόσες φορές θα εκπαιδευτεί το δίκτυο με τις διαθέσιμες εικόνες. Μεγάλη τιμή ελλοχεύει την υπερεκπαίδευση του δικτύου στις συγκεκριμένες εικόνες, ενώ μικρή τιμή θα υποεκπαιδεύσει το δίκτυο.

➤ **Steps_per_epoch:**

Καθορίζει τον αριθμό των batches που θα χρησιμοποιηθούν ανά εποχή. Μικρή τιμή σημαίνει πως θα χρησιμοποιούνται λίγα batches ανά εποχή ενώ η μέγιστη τιμή πρέπει να είναι τέτοια, που το γινόμενο batch_size*steps_per_epoch να μην υπερβαίνει το πλήθος των εικόνων, γιατί διαφορετικά θα ψάχνει για περισσότερες

εικόνες από όσες είναι διαθέσιμες. Επομένως, για να ξεπεραστεί αυτή η εξάρτηση (τουλάχιστον η άμεση εξάρτηση), αφήνω να επιλεγεί αυτόματα το καλύτερο με χρήση του `samples/Batch_size`.

Επιπλέον, χρησιμοποιώ και `callbacks` για να βελτιστοποιήσω την όλη διαδικασία. Στα `callbacks` επιτελούνται δύο διαδικασίες. Αρχικά, παρακολουθείται το δίκτυο, συγκεκριμένα παρακολουθούνται τα `misses` που κάνει στο `validation set`, και αποθηκεύονται τα βάρη τα οποία ελαχιστοποιούν το `validation_loss`. Δεύτερον, γίνεται έλεγχος αν το `validation_loss` παραμένει σταθερό, και έπειτα από έναν πεπερασμένο αριθμό `epoch`(5 στην προκειμένη περίπτωση) για τα οποία ισχύει η προϋπόθεση αυτή, σταματά η διαδικασία. Με αυτόν τον τρόπο, επιτυγχάνεται η χρήση των βέλτιστων βαρών, απλουστεύεται το δίκτυο, προκειμένου να αποφευχθεί το `over-fitting`, και μειώνεται ο χρόνος εκτέλεσης, αφού υπάρχει η δυνατότητα να μην εκτελεστούν όλες οι εποχές που επιλέχθηκαν.

Τέλος, χρησιμοποιώ `optimizer`, προκειμένου να εφαρμοστεί το `back-propagation` που χρειάζεται, με `learning rate` να έχει την `default` τιμή 0.001, ενώ ως `loss` ορίζω `cross-entropy`, διότι έχουμε περισσότερες από 1 κλάσεις. Για τις διάφορες παραμετροποιήσεις που θα κάνω, θα χρησιμοποιήσω 3 `optimizers`, τον `Adam`, `Adadelta` & `Adagrad`.

- **Τοπολογία των επιλεγμένων αρχιτεκτονικών:**

Επέλεξα να υλοποιήσω 2 διαφορετικές αρχιτεκτονικές αρχικά, μία βασισμένη σε συνελκτικό νευρωνικό δίκτυο, μία σε πλήρες συνδεδεμένο δίκτυο και τέλος, μία που προκύπτει από τον συνδυασμό των δύο. Παρόλο που δοκίμασα και `batch_normalization layer`, παρατήρησα πως η εκπαίδευση αργούσε αρκετά, οπότε επέλεξα να μην το χρησιμοποιήσω τελικά.

➤ **Αρχιτεκτονική βασισμένη σε Convolutional Layers μόνο:**

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(256, (5, 5), strides=(2, 2), activation='relu', input_shape=(100,100, 1), padding = 'same'),
    tf.keras.layers.Conv2D(256, (4, 4), strides=(2, 2), activation='relu', padding = 'same'),
    tf.keras.layers.Conv2D(128, (3, 3), strides=(1, 1), activation='relu', padding = 'same'),
    tf.keras.layers.Conv2D(128, (3, 3), strides=(1, 1), activation='relu', padding = 'same'),
    tf.keras.layers.MaxPooling2D(pool_size = (2, 2), strides=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(rate = 0.3),
    tf.keras.layers.Dense(34, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'])
model.summary()
```

Εικ.6 Η τοπολογία της 1^{ης} αρχιτεκτονικής (ενδεικτικές τιμές υπερπαραμέτρων και επιλογή `optimizer`).

Αποφάσισα να χρησιμοποιήσω 4 convolutional layers, με σκοπό την εξαγωγή ικανοποιητικού πλήθους χαρακτηριστικών από τις εικόνες, ενώ παράλληλα προσδίδω βάθος στο δίκτυο. Στην συνέχεια, ακολουθούνται από 1 layer Max-Pool για να μειωθούν οι διαστάσεις και οι υπολογιστικές ανάγκες του μοντέλου, 1 Flatten layer για να μετατρέψει σε διανύσματα τις εισόδους του, 1 Drop-Out layer με σκοπό να ελαχιστοποιήσει την επίδραση ενός πιθανού φαινομένου over-fitting και 1 Dense layer, προκειμένου να λειτουργήσει ως ταξινομητής και να κάνει την ταξινόμηση, όπου οι παράμετροί του είναι σταθεροί, καθώς το πλήθος των νευρώνων καθορίζεται από το πλήθος των κλάσεων που υπάρχουν (34 στο database) και με συνάρτηση ενεργοποίησης την *softmax*, με σκοπό να μετατραπεί η πρόβλεψη σε πιθανότητα για να προκύψει η κατηγοριοποίηση.

➤ Μια πλήρη αρχιτεκτονική ενός CNN:

```
#Complete CNN
model1 = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(256, (4, 4), strides=(2, 2), activation='relu', input_shape=(100,100, 1), padding = 'same'),
    tf.keras.layers.Conv2D(256, (3, 3), strides=(2, 2), activation='relu', padding = 'same'),
    tf.keras.layers.Conv2D(128, (3, 3), strides=(2, 2), activation='relu', padding = 'same'),
    tf.keras.layers.Conv2D(64, (2,2),strides=(1,1),activation='relu',padding='same'),
    tf.keras.layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),
    tf.keras.layers.Dropout(rate = 0.3),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(rate = 0.25),
    tf.keras.layers.Dense(34, activation='softmax')
])
model1.compile(optimizer=tf.keras.optimizers.Adam(),#or adamax
               loss = 'categorical_crossentropy',
               metrics = ['accuracy'])
```

Εικ.8 Η τοπολογία της 3^{ης} αρχιτεκτονικής (ενδεικτικές τιμές υπερπαραμέτρων και επιλογή optimizer).

Σε αυτό το δίκτυο, χρησιμοποιούνται ορισμένα convolutional layers στην αρχή για τους ίδιους λόγους με την προηγούμενη αρχιτεκτονική, ακολουθούμενα από ένα pool, ένα drop-out & ένα flatten layer. Στην συνέχεια, επέλεξα μερικά fully connected, ένα drop-out layer ξανά για να απλοποιηθεί η τελική επιλογή. Σε όλες τις αρχιτεκτονικές, το πλήθος των νευρώνων σε κάθε layer επιλέχθηκε ψευδοτυχαία. Συγκεκριμένα, επέλεξα να είναι δυνάμεις του 2, και τέτοιες τιμές, ώστε να μην είναι πολύ μικρές, με αποτέλεσμα η εκπαίδευση να γίνεται αργά και με χαμηλά ποσοστά απόδοσης, αλλά και να μην είναι πολύ μεγάλες, προκειμένου να μην υπερεκπαιδευτεί το δίκτυο στις συγκεκριμένες εικόνες, να μην εμφανιστεί φαινόμενο over-fitting και να μην χρειάζεται υπερβολικά μεγάλος χρόνος εκπαίδευσης. Η συνάρτηση ενεργοποίησης είναι η Relu, καθώς, έπειτα και από σχετική αναζήτηση, αποδίδει καλύτερα από τις υπόλοιπες με διαφορά, ενώ padding χρησιμοποιείται σε

όλα τα μοντέλα, προκειμένου να αποφευχθούν προβλήματα με τις διαστάσεις των εικόνων, καθώς αυτές διατρέχουν το δίκτυο.

- **Τιμές παραμέτρων/υπερπαραμέτρων – Σύγκριση αποτελεσμάτων**

Όπως είναι πολύ λογικό, οι τιμές των παραμέτρων είναι απεριόριστες, καθώς και η επιλογή των layers. Επέλεξα ως είσοδο των εικόνων, όπως αναφέρθηκε ήδη, το μέγεθος (100,100) και επομένως το batchsize στην μέγιστη τιμή του, τέτοια ώστε να μην προκύπτει OOM σφάλμα (Out Of Memory), λόγω του μεγάλου μεγέθους δεδομένων. Το μέγεθος με το οποίο θα διαβαστεί η εικόνα, αποτελεί σημαντικό παράγοντα καθώς κατά την σμίκρυνση ή μεγέθυνση της εικόνας (ανάλογα με τις αρχικές τις διαστάσεις) ελλοχεύει ο κίνδυνος να αλλάξουν τα χαρακτηριστικά που ξεχωρίζουν την κάθε εικόνα. Τέλος, τα παραπάνω ισχύουν και για τις 2 αρχιτεκτονικές, και παραμένουν σταθερά, ενώ δεν είναι απαραίτητο όταν γίνεται overfitting στην μία αρχιτεκτονική, να γίνεται και στην δεύτερη. Φυσικά, επειδή οι τιμές κατά την αρχικοποίηση των βαρών είναι τυχαίες, τα συμπεράσματα δεν είναι απόλυτα. Τέλος, οι τιμές των accuracy είναι προσεγγιστικές, με την έννοια ότι αν ξανατρέξει η εκπαίδευση και το Jupyter Notebook από την αρχή θα προκύψει accuracy κοντά στις παραπάνω τιμές.

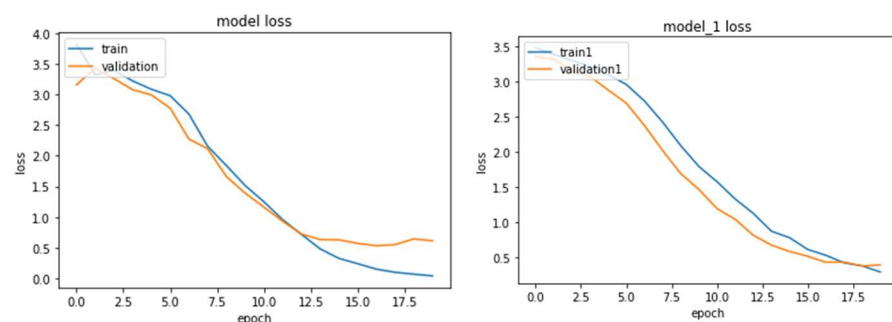
Training Batch Size	Validation Batch Size	Testing Batch Size	Μέγεθος εικόνων που διαβάζονται
800	700	1000	(100,100)

➤ **Επίδραση της τιμής των epoch που θα εκτελεστούν:**

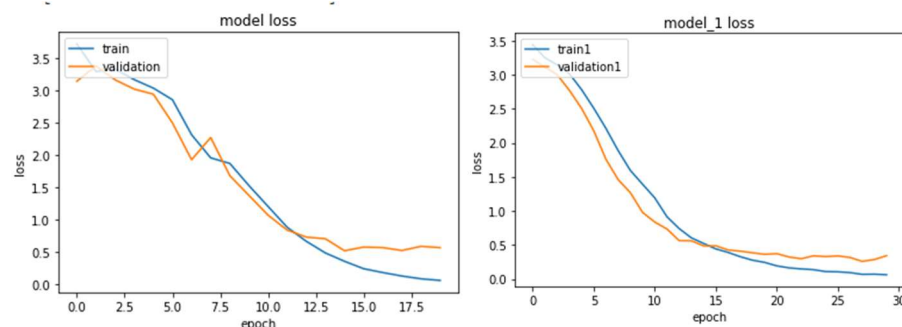
Η τιμή των epoch πρέπει να είναι τέτοια που να μην προκαλεί overfitting στο δίκτυο, με αποτέλεσμα αυτό να εκπαιδεύεται πολύ καλά στις εικόνες που χρησιμοποιούνται κατά την εκπαίδευση, αλλά ούτε και να έχει τέτοια τιμή, που να κάνει underfitting, με αποτέλεσμα να είναι υπό-εκπαιδευμένο το δίκτυο. Επίσης, καλό θα ήταν να έχει μια τιμή που να μην οδηγεί την διαδικασία σε χρονοβόρα διάρκεια εκπαίδευσης. Επομένως, επέλεξα αρχικά την τιμή 30, ενώ έκανα και 3 δοκιμές, μία με 20, μία με 40 και μία με 60. Ακολουθούν τα αποτελέσματα των δοκιμών, σύμφωνα με το model.evaluate και την χρήση του testing dataset.

Τιμή Epoch	Accuracy 1ης αρχιτεκτονικής	Accuracy 2ης αρχιτεκτονικής	Loss 1ης αρχιτεκτονικής	Loss 2ης αρχιτεκτονικής
20	0.9195	0.8986	0.4294	0.3895
30	0.9204	0.9558	0.3959	0.2197
40	0.9269	0.9521	0.3269	0.2496
60	0.9139	0.9395	0.3687	0.2746

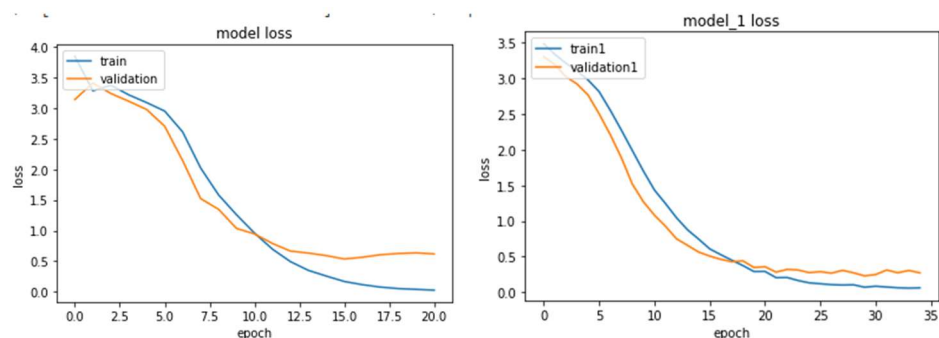
Παρατηρώ ότι το accuracy αυξάνεται, και στην συνέχεια αρχίζει να μειώνεται, αντί για να μειώνονται συνεχώς, επομένως αρχίζει να δημιουργείται φαινόμενο overfitting. Το γεγονός αυτό οφείλεται στο γεγονός ότι το πλήθος των εποχών είναι αρκετά μεγάλο, επομένως εκπαιδεύεται παραπάνω στις συγκεκριμένες εικόνες του training_set και επειδή οι εικόνες στο test_set μοιάζουν με αυτές του training, εμφανίζει αυτό το μεγάλο ποσοστό επιτυχίας, όμως σε περίπτωση που οι εικόνες ήταν διαφορετικές, θα υπήρχε χαμηλότερο ποσοστό επιτυχίας. Επίσης, με βάση τα παρακάτω διαγράμματα παρατηρώ πως σε ορισμένες φορές, με την χρήση των callbacks δεν τρέχουν καν οι 40 ή 60 εποχές, ενώ το validation_loss ορισμένες φορές τρεμοπαιζει, σημάδι ότι αρχίζει να γίνεται overfitting. Έτσι, αποφάσισα να κρατήσω τις 30 εποχές σταθερές για τις επόμενες δοκιμές.



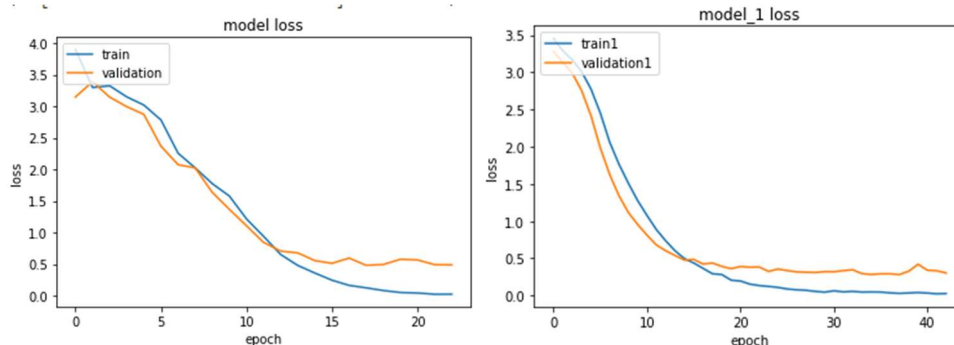
Εικ.9,10 Η μείωση του loss για training και validation των 2 μοντέλων για 20 εποχές(αριστερά η 1^η,δεξιά η 2^η).



Εικ.11,12 Η μείωση του loss για training και validation των 2 μοντέλων για 30 εποχές(αριστερά η 1^η,δεξιά η 2^η).



Εικ.13,14 Η μείωση του loss για training και validation των 2 μοντέλων για 40 εποχές(αριστερά η 1^η,δεξιά η 2^η).



Εικ.15,16 Η μείωση του loss για training και validation των 2 μοντέλων για 60 εποχές(αριστερά η 1^η,δεξιά η 2^η).

➤ Επίδραση της χρήσης **callbacks**:

Callbacks	Accuracy 1 ^{ης} αρχιτεκτονικής	Accuracy 2 ^{ης} αρχιτεκτονικής
Με callbacks	0.9204	0.9558
Χωρίς callbacks	0.9037	0.9376

Άμα δεν χρησιμοποιηθούν τα **callbacks**,σημαίνει πως το δίκτυο θα συνεχίσει να εκπαιδεύεται, ανεξάρτητα από το άμα βελτιώνεται το `val_loss` ή όχι. Επομένως, ελλοχεύει και πάλι ο κίνδυνος του **overfitting**, καθώς και το να απομακρυνθεί το δίκτυο από τα βέλτιστα βάρη. Πράγματι, με βάση τα παραπάνω αποτελέσματα επαληθεύονται αυτά που περίμενα.

➤ Επίδραση του πλήθους των **νευρώνων**:

Η επιλογή του πλήθους των νευρώνων, όπως αναλύθηκε στην αντίστοιχη υποενότητα, είναι εντελώς τυχαία. Φυσικά, όσο αυξάνεται ο αριθμός των νευρώνων, αυξάνεται και ο χρόνος εκτέλεσης, αφού πρέπει να υπολογιστούν περισσότεροι παράγοντες. Επίσης, δεν είναι απαραίτητο πως θα αυξηθεί η απόδοση, καθώς υπάρχει περίπτωση να υπερεκπαιδευτεί το δίκτυο στις εικόνες του training set.

Layer (type)	Output Shape	Param #
conv2d_183 (Conv2D)	(None, 50, 50, 256)	6656
conv2d_184 (Conv2D)	(None, 25, 25, 256)	1048832
conv2d_185 (Conv2D)	(None, 25, 25, 128)	295040
conv2d_186 (Conv2D)	(None, 25, 25, 128)	147584
max_pooling2d_45 (MaxPooling)	(None, 12, 12, 128)	0
flatten_44 (Flatten)	(None, 18432)	0
dropout_49 (Dropout)	(None, 18432)	0
dense_92 (Dense)	(None, 34)	626722
Total params: 2,124,834		
Trainable params: 2,124,834		
Non-trainable params: 0		

Layer (type)	Output Shape	Param #
conv2d_175 (Conv2D)	(None, 50, 50, 256)	4352
conv2d_176 (Conv2D)	(None, 25, 25, 256)	590080
conv2d_177 (Conv2D)	(None, 13, 13, 128)	295040
conv2d_178 (Conv2D)	(None, 13, 13, 64)	32832
max_pooling2d_43 (MaxPooling)	(None, 6, 6, 64)	0
dropout_46 (Dropout)	(None, 6, 6, 64)	0
flatten_42 (Flatten)	(None, 2304)	0
dense_88 (Dense)	(None, 256)	590080
dense_89 (Dense)	(None, 128)	32896
dropout_47 (Dropout)	(None, 128)	0
dense_90 (Dense)	(None, 34)	4386
Total params: 1,549,666		
Trainable params: 1,549,666		
Non-trainable params: 0		

Εικ.17,18 Οι παράμετροι που προκύπτουν μετά από κάθε layer και το σύνολό τους.(Αριστερά η 1^η αρχιτεκτονική, δεξιά η 2^η)

➤ Επίδραση των **kernel size, stride & pool size**:

Οι πυρήνες και τα stride επηρεάζουν το πόσο μεγάλο ή μικρό είναι το μέρος της εικόνας που δέχονται ως είσοδο οι νευρώνες. Ειδικά για τα συνελκτικά layer, το μέγεθος των φίλτρων επιδρά στην έξοδο τους, καθώς το τελικό αποτέλεσμα προκύπτει από την συνέλιξη του φίλτρου με το μέρος που βλέπει ο πυρήνας.

Επομένως, για μεγάλες τιμές του πυρήνα θα γίνεται μεγαλύτερη συνέλιξη μεταξύ των στοιχείων της εικόνας και του πυρήνα, και με μεγαλύτερο strides, θα μετατοπίζεται σε μεγαλύτερη απόσταση μετά από κάθε συνέλιξη. Έτσι, υπάρχει περίπτωση να μην γίνεται καλή διάκριση των χαρακτηριστικών κάθε περιοχής. Επίσης, μεγάλη τιμή strides σημαίνει πως η εικόνα θα διατρέχεται πιο γρήγορα, άρα θα εξοικονομείται χρόνος. Τέλος, μεγάλο pool_size σημαίνει πως θα υπάρχει μεγαλύτερο εύρος τιμών οι οποίες θα υποδειγματοληπτούνται, επομένως θα αυξάνεται το εύρος της υποδειγματοληψίας και άρα υπάρχει κίνδυνος να απορρίπτονται πληροφορίες.

Μέγεθος Πυρήνα	Accuracy 1ης αρχιτεκτονικής	Accuracy 2ης αρχιτεκτονικής
(3,3)	0.919	0.9432
(5,5)	0.9246	0.92
Μέγεθος Strides	Accuracy 1ης αρχιτεκτονικής	Accuracy 2ης αρχιτεκτονικής
(2,2)	0.8781	0.9507
(3,3)	0.5537	0.6198

Συμπέρανα πως η καλύτερη λύση είναι η χρήση διάφορων μεγεθών πυρήνων και strides και όχι ένα μέγεθος για όλα, επομένως στο τελικό δίκτυο υλοποιείται αυτό ακριβώς.

➤ Επίδραση της επιλογής **optimizer**:

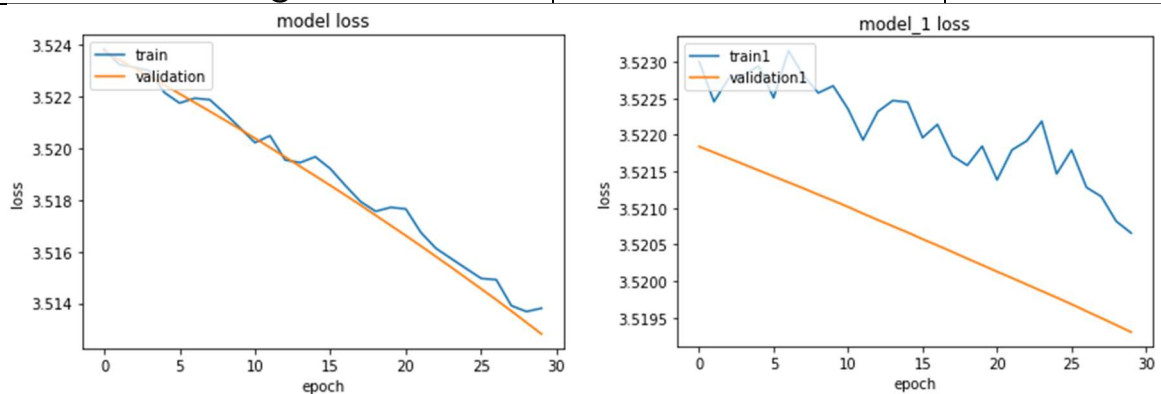
Η κύρια διαφορά που υπάρχει μεταξύ στις διάφορες επιλογές optimizer είναι ο τρόπος που υπολογίζουν και μεταβάλλουν το learning rate τους, τον τρόπο ελαχιστοποίησης του loss και τον υπολογισμό των βαρών του κάθε νευρώνα. Μερικοί από αυτούς είναι οι:

- Adagrad: Κάθε στοιχείο διαιρείται με την τετραγωνική ρίζα του ανά-στοιχείου άθροισμα των τετραγώνων των παραγώγων.

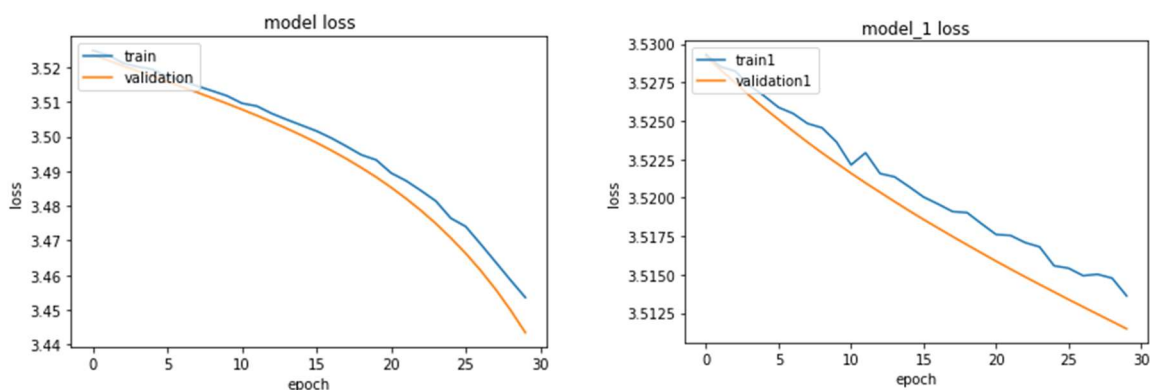
- Adadelta: όπου το άθροισμα αντικαθίσταται με ένα ολοένα και μικρότερο άθροισμα, που προκύπτει από τις αλλαγές των παραμέτρων.
- Adam: ο πιο κοινός optimizer, αποτελεί συνδυασμός των παραπάνω και μερικών ακόμα optimizers.

Επομένως, η κατάλληλη επιλογή optimizer αποτελεί σημαντικό παράγοντα, καθώς ο κάθε optimizer υπολογίζει και μεταβάλλει διαφορετικά το learning rate.

Είδος optimizer	Accuracy 1ης αρχιτεκτονικής	Accuracy 2ης αρχιτεκτονικής
Adam	0.9204	0.9265
Adadelta	0.1661	0.0279
Adagrad	0.2457	0.0935



Εικ.17,18 Η διακύμανση του val_loss και του loss κατά την διάρκεια της εκπαίδευσης με Adadelta Optimizer(αριστερά η 1^η αρχιτεκτονική ,δεξιά η 2^η)



Εικ.19,20 Η διακύμανση του val_loss και του loss κατά την διάρκεια της εκπαίδευσης με Adagrad Optimizer(αριστερά η 1^η αρχιτεκτονική ,δεξιά η 2^η)

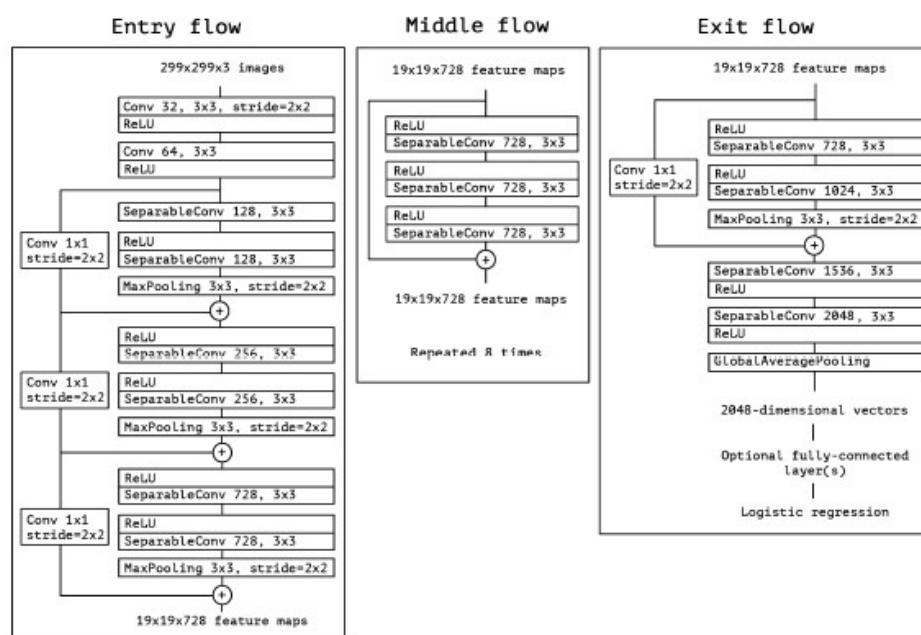
Με βάση τα παραπάνω αποτελέσματα και με τα διαγράμματα, παρατηρώ πως αντί το loss να μειώνεται πιο απότομα κατά την εκπαίδευση, εμφανίζει μια γραμμικότητα, που ενδεχομένως οφείλεται στον τρόπο υπολογισμού των losses και του learning rate. Επίσης, οδηγούμαι στο συμπέρασμα πως αν ήταν περισσότερος ο αριθμός των εποχών να ήταν πιο ικανοποιητικά τα αποτελέσματα και τελικά να επιτυγχανόταν το επιθυμητό

αποτέλεσμα. Επομένως, η κατάλληλη επιλογή είναι ο optimizer Adam, αφού χρειάζεται λιγότερο χρόνο για να παράξει ικανοποιητικά αποτελέσματα.

2. Προ-εκπαιδευμένο δίκτυο:

Οι 2 αρχιτεκτονικές που επέλεξα είναι η Xception και η InceptionResNetV2, όπου τις επέλεξα με βάση το πόσο μεγάλες είναι, με την Xception να είναι σχετικά μικρή(88mb) και την InceptionResNetV2 να είναι μεσαίου μεγέθους(215 mb), όπου το μέγεθος είναι ανάλογο με το πλήθος των νευρώνων και του βάθους σε κάθε δίκτυο, χωρίς τις δικές μου παρεμβολές σε layers.

- **1^η αρχιτεκτονική: Xception**



Εικ.21 Το βασικό Xception μοντέλο.

```
model.add(layers.Dense(32,activation='relu'))
model.add(layers.Dropout(rate = 0.4))
model.add(layers.Flatten())
model.add(layers.Dropout(rate = 0.3))
model.add(layers.Dense(34, activation='softmax'))

model.summary()
model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.Adam(),
              metrics=['acc'])
```

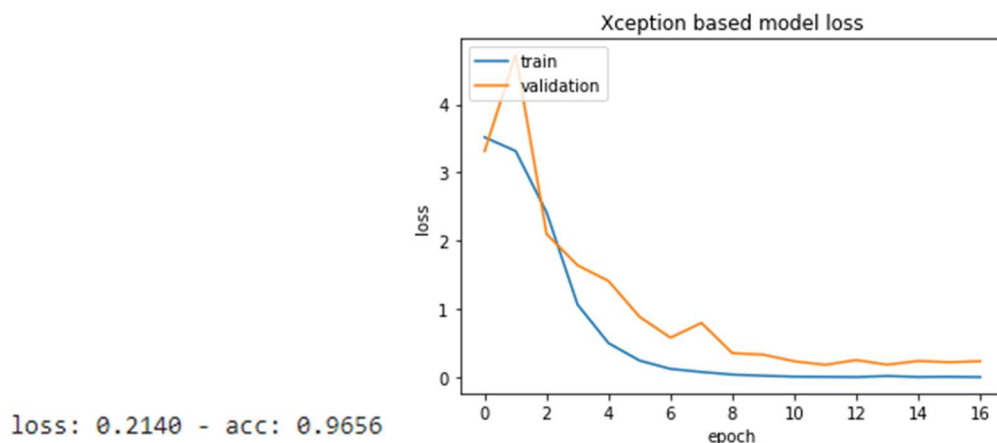
Εικ.22 Η παραμετροποίηση, χρησιμοποιώντας ως βάση το Xception μοντέλο.

Η βάση του δικτύου είναι η χρήση μόνο συνελίξεων(και των επακόλουθων layers), η οποίες γίνονται κατά βάθος και διακριτές μεταξύ τους. Έτσι, χρησιμοποιούνται 36 Convolutional Layers, για να παραχθούν τα feature maps, όπου μεταξύ μερικών Convolutions

υπάρχει layer Max-Pooling.Επομένως διαμορφώνεται ένα δίκτυο με διακριτές συνδέσεις μεταξύ των layers.Ωστόσο, προκειμένου να αυξηθεί σε ικανοποιητικό βαθμό η απόδοση της αρχιτεκτονικής πρόσθετα ένα μικρό dense layer με 32 νευρώνες και συνάρτησης ενεργοποίησης την Relu,ακολουθούμενο από ένα drop-out layer με σχετικά μεγάλο rate = 0.4,ενώ τέλος υπάρχει ένα flatten και ένα ακόμα drop-out layer με rate = 0.3 για να ακολουθήσει το dense layer των 34 κλάσεων με συνάρτηση ενεργοποίησης την softmax για να κάνει την ταξινόμηση. Επιπρόσθετα, ορίζω έναν ακόμα optimizer για να χρησιμοποιηθεί, τον Adam.

➤ Υπερπαράμετροι δικτύου:

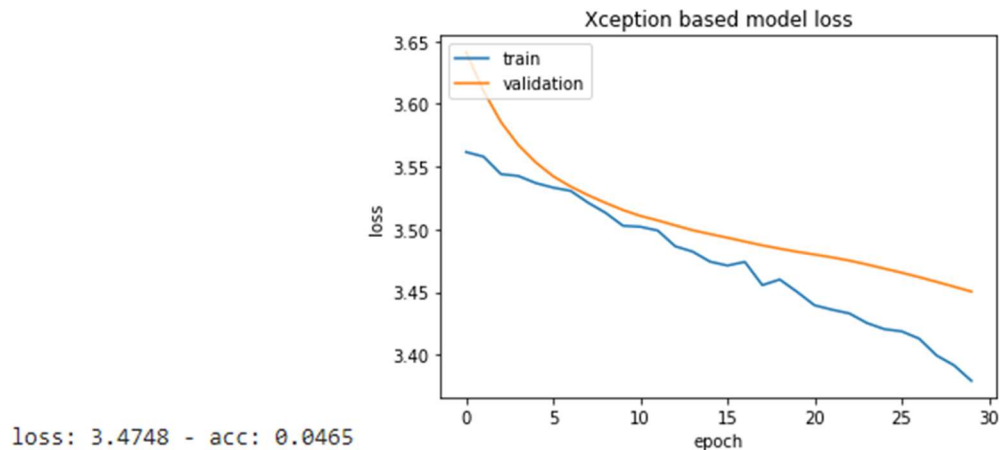
Έχοντας ως δεδομένο ότι υπάρχουν εκατομμύρια νευρώνες, θεώρησα ότι η εκπαίδευση θα γίνει σχετικά γρήγορα, επομένως επέλεξα 20 εποχές. Επίσης, λόγω του μεγάλου μεγέθους του δικτύου και των πόρων που χρειάζεται, το batch_size είναι μικρότερο από πριν, ενώ διατήρησα τις διαστάσεις της εικόνας στις ίδιες τιμές με πριν (100,100).Επίσης χρησιμοποιώ callbacks για να μην ξεφύγει η εκπαίδευση και υπάρχει overfitting, με την τιμή patience να ορίζεται στα 5, δηλαδή αν για 5 εποχές το val_loss που προκύπτει ξεπερνά κάποιο val_loss των προηγούμενων εποχών, σταματά νωρίτερα η διαδικασία. Τέλος, ως steps_per_epoch & validation_step όρισα τα αντίστοιχα samples / των αντίστοιχων batch_sizes.Τα αποτελέσματα που προκύπτουν αποτυπώνονται στις παρακάτω εικόνες.



Εικ.23,24 Η ευστοχία του τελικού μοντέλου και η διακύμανση του val_loss κατά την εκπαίδευση.

Παρατηρώ ένα overshoot, κοντά στην αρχή, το οποίο οφείλεται στο ότι η εκπαίδευση δεν ξεκινάει με τυχαία βάρη όπως προηγουμένως, αλλά με βάρη από το πρόβλημα του ImageNet, επομένως χρειάζονται κάποιες εποχές προκειμένου τα βάρη να προσαρμοστούν στο τρέχον πρόβλημα. Επιπλέον, το overshoot δεν είναι σίγουρο ότι θα συμβεί, ωστόσο όταν τραβήχτηκε το

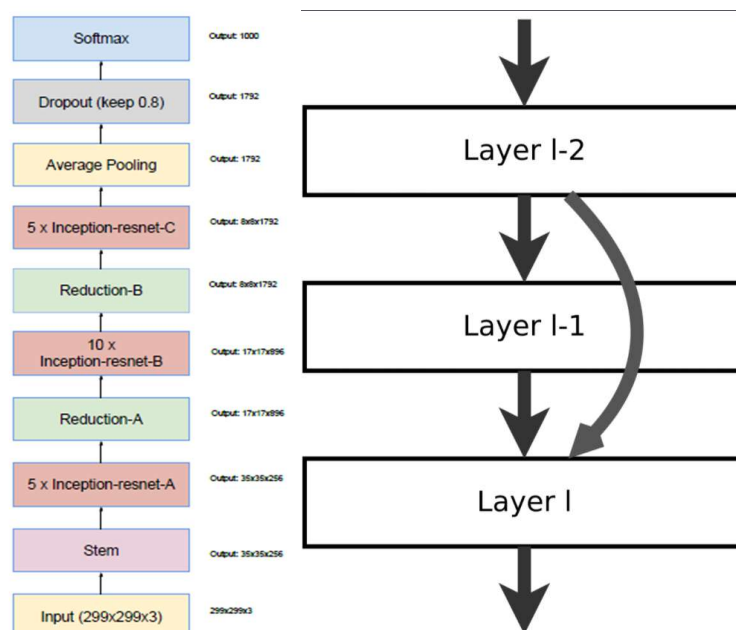
παραπάνω screenshot εμφανίστηκε. Επίσης, παρατηρώ ότι με χρήση των callbacks, η εκπαίδευση σταμάτησε πιο νωρίς, στην 16^η-17^η εποχή. Τέλος, σε σύγκριση με πριν, το αποτέλεσμα είναι σαφώς καλύτερο αφού το accuracy αυξήθηκε τουλάχιστον κατά 2%. Ακολουθούν τα αποτελέσματα αλλάζοντας διάφορες παραμέτρους, όπως τον optimizer, τα layers που προστέθηκαν, την μη χρήση callback, την αύξηση των εποχών κ.α.



Εικ.25,26 Τα αποτελέσματα για χρήση διαφορετικών τιμών στις διάφορες παραμέτρους του δικτύου.

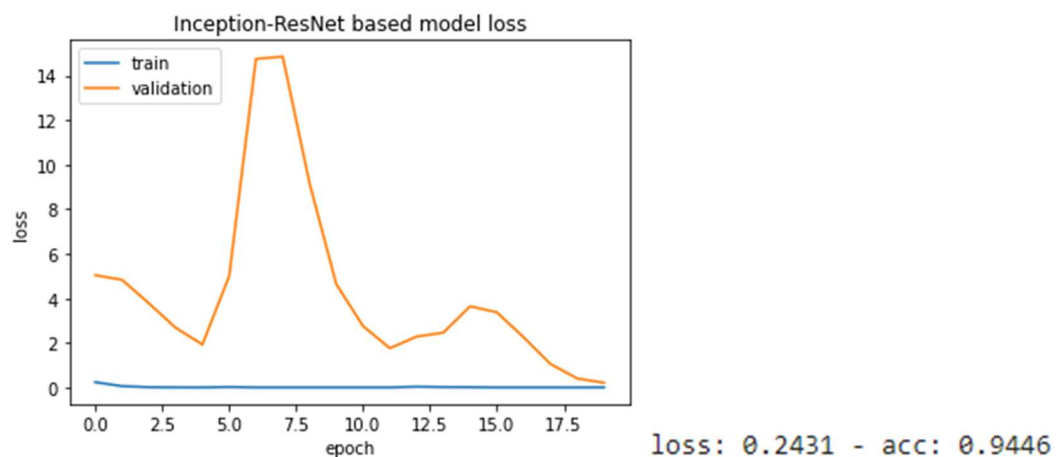
Με βάση τα παραπάνω ενδεικτικά αποτελέσματα, οι αλλαγές που έκανα είχαν καταστρεπτικές συνέπειες στην ευστοχία του δικτύου, όπως φαίνεται από το acc κατά το testing αλλά και από την διακύμανση του val_loss κατά την διάρκεια του training. Φυσικά, όλες οι τιμές που χρησιμοποίησα είναι ενδεικτικές και υπάρχουν τιμές για τις οποίες το δίκτυο είναι πιο εύστοχο, πιο αργό/γρήγορο κ.τ.λ.

- **2^η αρχιτεκτονική: InceptionResNetV2**



Εικ.27,28 Η βασική τοπολογία του InceptionResNetV2 (αριστερά), ενώ δεξιά απεικονίζεται ένα παράδειγμα ResNet δικτύου.

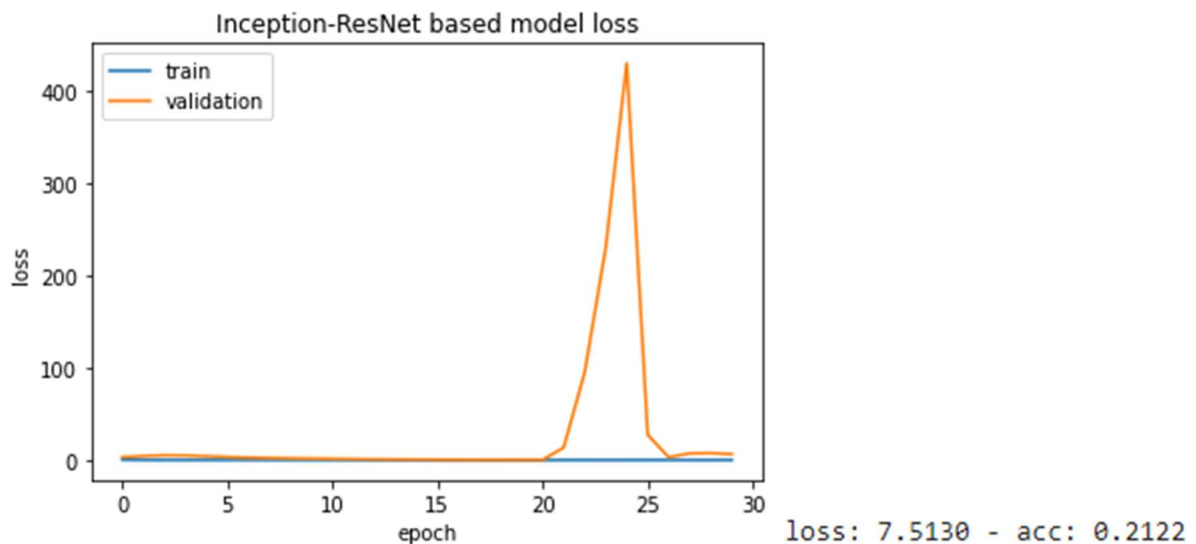
Ως Residual Network(ResNet) ορίζεται το δίκτυο του οποίου τα layers, μέσω συντομεύσεων ή υπερπηδήσεων, παραλείπουν επόμενα layers ή συνδέονται με layers που εντοπίζονται παρακάτω και έτσι δεν είναι απαραίτητο να δημιουργήσει γραμμικές συνδέσεις μεταξύ των layers. Στο παράδειγμα της εικόνας, το layer l-2 εμφανίζει συνδέσεις και με το επόμενο layer l-1 αλλά και με το μεθεπόμενο layer l. Εφόσον κάποιες συνδέσεις μπορούν να παραλειφθούν, επιταχύνεται η διαδικασία της εκπαίδευσης. Όλα τα κομμάτια της τοπολογίας(Stem, Reduction-A, Inception-resnet-A,...) περιέχουν μια μίξη Convolutional layers με Pooling Layers(είτε Max είτε Average), όπου οι υπερπαραμέτροί τους παίρνουν διάφορες τιμές, ωστόσο θεωρώ πως η ανάλυση του κάθε κομματιού ξεχωριστά ξεφεύγει από την παρούσα αναφορά. Ακολουθούν τα πρόσθετα layer που εισήγαγα στην αρχιτεκτονική, όπου τοποθετώ ένα drop-out layer με σκοπό να απλοποιήσω την όλη αρχιτεκτονική, όπου υπάρχουν κοντά στους 55.000.000 νευρώνες, ένα flatten, ένα ακόμα drop-out layer για τους ίδιους λόγους με το προηγούμενο και το τελευταίο είναι ένα fully connected layer, με 34 νευρώνες για την ταξινόμηση. Χρησιμοποίησα 20 εποχές, και τις ίδιες ρυθμίσεις με το Xception δίκτυο, και στις εικόνες που ακολουθούν απεικονίζονται τα αποτελέσματα για μία συγκεκριμένη εκτέλεση του κώδικα.



Εικ.29,30 Τα αποτελέσματα με την χρήση του InceptionResNetV2 μοντέλου.

Από τα παραπάνω, παρατηρώ πως εμφανίζονται και στην συγκεκριμένη υλοποίηση over-shoots, δηλαδή εκτινάξεις του val_loss, ωστόσο στην συνέχεια μειώνεται και τελικά υπάρχει καλή απόδοση στο δίκτυο, λίγο μικρότερη από αυτήν του Xception μοντέλου. Έπειτα, άλλαξα το batchsize(έγινε 800),αφαίρεσα ένα layer(το 1^ο dropout) και αύξησα τις εποχές από 20 σε 30, για να μπορέσω να δω πως θα αλλάξουν τα παραπάνω

νούμερα. Φυσικά, λόγω της αύξησης των εποχών και του batch_size αυξάνεται ο χρόνος εκτέλεσης της εκπαίδευσης σε σχέση με πριν. Παρατήρησα, και σε συνδυασμό με το παρακάτω διάγραμμα και με το accuracy που προέκυψε, για την συγκεκριμένη εκτέλεση, ότι και σε αυτήν την περίπτωση, το accuracy μειώθηκε, παρόλο που το val_loss απέκτησε τελικά χαμηλή τιμή. Αυτό ενδεχομένως να οφείλεται στο γεγονός ότι “έτρεξαν” 30 εποχές(ή λίγο λιγότερες), οπότε έγινε overfitting στο δίκτυο.



Εικ.31,32 Τα αποτελέσματα για χρήση διαφορετικών τιμών στις διάφορες παραμέτρους του δικτύου.

• Βιβλιογραφία

- **Richard Szeliski**: Computer Vision Algorithms and Applications
- Keras Documentation: <https://keras.io/api/>
- Wikipedia : <https://en.wikipedia.org/>
- **Arcangelo Distanto, Cosimo Distanto**: Handbook of Image Processing and Computer Vision, Volume 3: From Pattern to Object
- **Nikhil Buduma, Nicholas Locascio** : Fundamentals of Deep Learning, Designing Next-Generation Machine Intelligence Algorithms
- **Yoshua Bengio, Aaron Courville, Ian Goodfellow** - Deep learning
- **Francois Chollet**: Xception: Deep Learning with Depthwise Separable Convolutions
- **Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi**: Inception-v4, Inception-Resnet and the Impact of Residual Connections on Learning