Ατομική Τελική εργασία

Καραπέπερα Ελπίδα
57423

ΑΝΑΓΝΩΡΙΣΗ ΠΡΟΤΥΠΩΝ

Sign Language Digits Dataset

ΠΕΡΙΕΧΟΜΕΝΑ

1. Απλό Τεχνητό Νευρωνικό Δίκτυο / Artificial Neural Network (ANN) 2
a.....3
b.....5
c.....10
d.....17
e.....18

2. Συνελικτικά Νευρωνικά Δίκτυα / Convolutional Neural Networks (CNN) 23
a.....23
b.....24
c.....27
d.....29
e.....34

3. Ερωτήσεις Κατανόησης.....37
a.....37
b.....37

4. Bonus 39
a.....39
b.....41
c.....41

Artificial Neural Network

Αρχικά χωρίστηκε το dataset σε train και validation sets. Τα στοιχεία που περιέχει το κάθε dataset επιλέγονται τυχαία μέσα από το dataset. Συγκεκριμένα, επιλέγονται τυχαία το 70% των στοιχείων του dataset για να χρησιμοποιηθούν στο train dataset. Στη συνέχεια γίνεται shuffle και επιλέγεται τυχαία από τα στοιχεία το 20% για να χρησιμοποιηθεί στο validation dataset και τα στοιχεία του αρχικού dataset ξανά ανακατεύονται (shuffle). Το test dataset δημιουργείται στο τέλος, για τη δημιουργία του οποίου επιλέγεται το 10% των στοιχείων του συνολικού dataset.

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.3,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size,
  shuffle="true",
  color_mode="grayscale")
```

Δημιουργία train dataset

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size,
  shuffle="true",
  color_mode="grayscale")
```

Δημιουργία validation dataset

```
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.1,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size,
  shuffle="true",
  color_mode="grayscale")
```

Δημιουργία test dataset

Είναι λίγο περίεργος ο τρόπος δημιουργίας του test dataset αλλά δουλεύει. Ο μόνος άλλος τρόπος δημιουργίας του ήταν να γίνει χειροκίνητα η δημιουργία κάθε dataset επιλέγοντας τυχαία έναν αριθμό εικόνων από κάθε ομάδα (‘0’, ‘1’, ‘2’, …, ‘9’) και τοποθετώντας το στο αντίστοιχο set (training, validation, test) ενώ ταυτόχρονα σε ένα πίνακα αποθηκεύεται και η κλάση στην οποία ανήκει η κάθε εικόνα του κάθε σετ. Θεώρησα ότι δεν υπάρχει λόγος για αυτή τη διαδικασία (δεν ήταν αυτή η ουσία της εργασίας) και επέλεξα αυτή τη γρήγορη και εύκολη διαδικασία που δημιουργεί ουσιαστικά ένα training dataset που έχει μέγεθος 70% του μεγέθους του συνολικού dataset και 2 validation datasets που το ένα έχει μέγεθος ίσο με το 20% του συνολικού και το άλλο 10% του συνολικού. Το πρώτο validation dataset (20%) το χρησιμοποίησα κανονικά ως validation dataset και το δεύτερο (10%) το χρησιμοποίησα για test dataset.

Οι εικόνες επιλέχθηκε να διαβάζονται σε grayscale καθώς το χρώμα δε περιέχει καμία χρήσιμη πληροφορία (όπως θα είχε πχ στο διαχωρισμό λουλουδιών όπου κάθε είδος έχει συγκεκριμένο χρώμα πέρα από σχήμα). Αντιθέτως, διαφορές στο φωτισμό και στο χρώμα του χεριού μπορεί να μπερδέψουν το δίκτυο.

a. Υλοποιήστε ένα απλό τεχνητό νευρωνικό δίκτυο (fully connected) με δύο κρυφά επίπεδα 256 και 128 νευρώνων αντίστοιχα. Προσαρμόστε το επίπεδο εισόδου (input layer) και το επίπεδο εξόδου (output layer) κατάλληλα για το dataset που πρέπει να χρησιμοποιήσετε.

Το δίκτυο αποτελείται από 2 κρυφά επίπεδα: ένα με 256 νευρώνες και ένα με 128 νευρώνες.

Αρχικά, επειδή τα δεδομένα έχουν 3 διαστάσεις (φωτεινότητα, και διδιάσατος πίνακας) πρέπει να τα κάνουμε flatten ώστε να περάσει η πληροφορία σε μία μόνο διάσταση. Στη συνέχεια προστίθενται τα κρυφά επίπεδα και το output layer και το δίκτυο εκπαιδεύεται.

```
model = tf.keras.Sequential([
  layers.Flatten(),
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.Dense(units=256),
  layers.Dense(units=128),
  layers.Dense(units=num_classes)
])
```

```
model.compile(
  optimizer='adam',
  loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])
```

```
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=100,
  callbacks=[checkpoint, early]
)

# Save the model
model.save('last.h5')
```

Στο δίκτυο δίνονται 100 epochs για να εκπαιδευτεί.

Το batch size τίθεται σε 32 εικόνες.

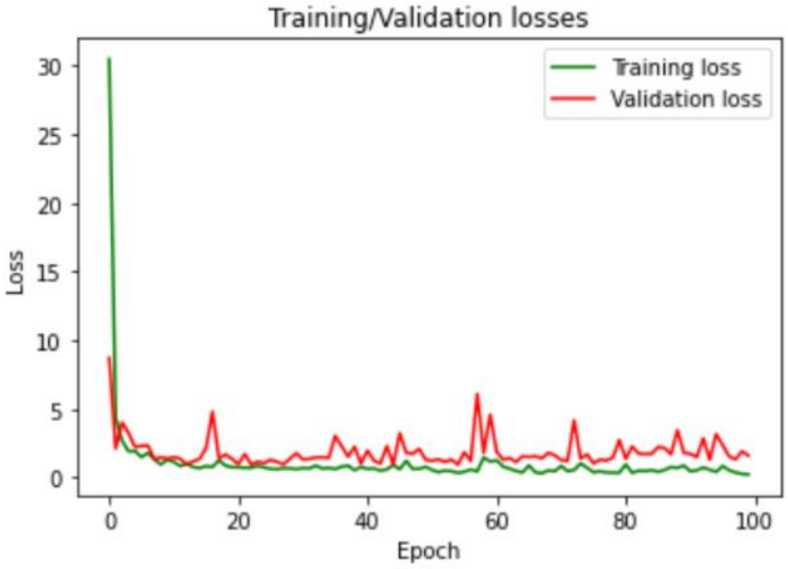
```
img_height = 100
img_width  = 100
classes    = ['0','1','2','3','4','5','6','7','8','9']
batch_size = 32
```

Ως optimizer χρησιμοποιήθηκε ο adam, ο οποίος είναι από τους πιο συνήθης optimizers καθώς μπορεί να διαχειριστεί πολύ καλά noisy συστήματα.

b. Παραγωγή των γραφημάτων train/test – loss ανά εποχή. Εμφανίζεται το πρόβλημα underfitting/overfitting; Αν ναι, να προτείνετε έναν τρόπο διαχείρισης του φαινομένου. Επίσης να αιτιολογήσετε ποιος είναι ο βέλτιστος αριθμός εποχών εκπαίδευσης. Για τον βέλτιστο αριθμό εποχών εκπαίδευσης, ζητείται να εξαχθούν οι μετρικές: accuracy, precision recall, f1 score, confusion matrix.

Overfitting συμβαίνει όταν το μοντέλο εκπαιδεύεται υπερβολικά πολύ πάνω σε συγκεκριμένα χαρακτηριστικά του training set με αποτέλεσμα, όταν δε βρίσκει τα ίδια στο validation set, να μην βάζει τα data points στην σωστή κλάση. (πχ τα χέρια στο training set έχουν μία συγκεκριμένη κλίση ή οπτική γωνία και αντί να αρκεστεί το μοντέλο στον αριθμό των δακτύλων που δείχνονται (έστω 1 για τον αριθμό 1) για την κατάταξη ενός data point στη συγκεκριμένη κλάση ζητάει το χέρι της εικόνας να έχει και την πολύ συγκεκριμένη κλίση (δεν αρκείται στο ότι δείχνει 1 μόνο δάχτυλο) και το απορρίπτει από την κλάση).

Underfitting είναι το αντίθετο του overfitting. Το δίκτυο έχει εκπαιδευτεί ελάχιστα και δεν έχει προλάβει να επιλέξει όλα τα χαρακτηριστικά που χρειάζονται για να κατατάξει ένα data point σε μία κλάση, με αποτέλεσμα να κατατάσσει σε αυτή και data points που δεν ανήκουν εκεί (πχ για την κατάταξη στον αριθμό 1 το δίκτυο μπορεί να αρκείται στην αναγνώριση ότι το αντικείμενο που απεικονίζεται βρίσκεται στο κέντρο της εικόνας).



Παρατηρούμε ότι αρχικά το validation loss είναι μικρότερο από το training loss. Αυτό συμβαίνει γιατί το training loss υπολογίζεται στην αρχή κάθε εποχής (πριν την αναδιαμόρφωση των βαρών των νευρώνων) ενώ το validation loss υπολογίζεται στο τέλος κάθε εποχής (αφού έχει βελτιωθεί λίγο το δίκτυο). Στη συνέχεια φαίνεται τα δύο errors να εξισώνονται και έπειτα σιγά σιγά το validation error να ξεπερνάει το training error. Αυτό είναι φυσικό εν μέρει αφού το δίκτυο εκπαιδεύεται πάνω στο training dataset, το validation dataset υπάρχει μόνο για τον έλεγχο, το γεγονός, όμως, ότι η διαφορά αυτή αυξάνεται είναι ένδειξη overfitting. Καθώς το δίκτυο εξειδικεύεται τόσο πολύ στις εικόνες του training set που δεν αναγνωρίζει ότι αυτές που ανήκουν στην ίδια κλάση του validation set πρέπει να ταξινομηθούν στην κλάση αυτή. Ιδανικά, για να μην υπάρχει overfitting η εκπαίδευση του δικτύου θα σταματούσε όταν το training και το validation error ήταν περίπου ίσα.

Το accuracy που επιτεύχθηκε ήταν 50.97%.

```
Found 2062 files belonging to 10 classes.
Using 206 files for validation.
7/7 [==============================] - 0s 6ms/step - loss: 1.6734 - accuracy: 0.5097
Test score:  1.6733765602111816
Test accuracy:  0.5097087621688843
```

Για τη διαχείριση του overfitting προτείνεται η μείωση των epochs στις οποίες εκπαιδεύεται το σύστημα.

Χρησιμοποιούνται early stoppings έτσι ώστε μετά από 20 epochs στα οποία δε θα παρατηρηθεί βελτίωση στην απόδοση του μοντέλου να σταματήσει το παραπάνω να εκπαιδεύεται. Αυτό γίνεται για οικονομία χρόνου. Επιλέγονται οι παρακάτω παράμετροι:

- monitor=’val_accuracy’ ώστε να αποθηκεύεται το accuracy που επιτυγχάνεται στο validation set σε κάθε epoch.
- verbose = 1 ώστε να φαίνεται αναλυτικά η πρόοδος σε κάθε epoch.
- min_delta=0 και patience=20 ώστε αν για 20 συνεχόμενα epochs δεν υπάρξει καμία βελτίωση να σταματήσει να εκπαιδεύεται το δίκτυο.

Τα checkpoints αποθηκεύουν κάθε φορά το epoch με το καλύτερο αποτέλεσμα ώστε να κρατηθεί το καλύτερο αποτέλεσμα που έχει επιτευχθεί. Επιλέγονται οι παρακάτω παράμετροι:

- monitor=’val_loss’ ώστε να αποθηκεύεται το loss του validation set σε κάθε epoch. Αυτό ουσιαστικά προσπαθούμε να ελαχιστοποιήσουμε γι’ αυτό και mode=’auto’.
- verbose = 1 ώστε να φαίνεται αναλυτικά η πρόοδος σε κάθε epoch.
- save_best_only=’True’ ώστε να αποθηκευτεί μόνο το καλύτερο αποτέλεσμα.
- save_freq=’epoch’ ώστε να αποθηκεύονται όλα τα epochs.

```
filepath = "weights.best.hdf5"
checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath, monitor='val_loss', verbose=1,
                             save_best_only=True, save_weights_only=False,
                             mode='auto', save_freq='epoch')

early = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=20,
                      verbose=1, mode='auto')
```

Όπως φαίνεται παρακάτω το δίκτυο παύει να εκπαιδεύεται στα 23 epochs. Δημιουργείται σε αυτό το σημείο το test dataset και γίνεται το evaluation του μοντέλου, όπου φαίνεται φανερά ότι υπάρχει overfitting, αυτό όμως θα παρατηρηθεί και στη συνέχεια στο ερώτημα b.

Κρατώντας, λοιπόν, το μικρότερο validation loss που έχουμε επιτύχει από όλα τα epochs είναι πλέον 66.5% (καλύτερο από το προηγούμενο: 50.97%).

```
Epoch 00063: early stopping
```

```
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.1,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size,
  shuffle="true",
  color_mode="grayscale")

model.load_weights("weights.best.hdf5")
score = model.evaluate(test_ds)
print('Test score: ', score[0])
print('Test accuracy: ', score[1])
```

```
Found 2062 files belonging to 10 classes.
Using 206 files for validation.
7/7 [==============================] - 0s 6ms/step - loss: 1.1054 - accuracy: 0.6650
Test score:  1.1053764820098877
Test accuracy:  0.6650485396385193
```

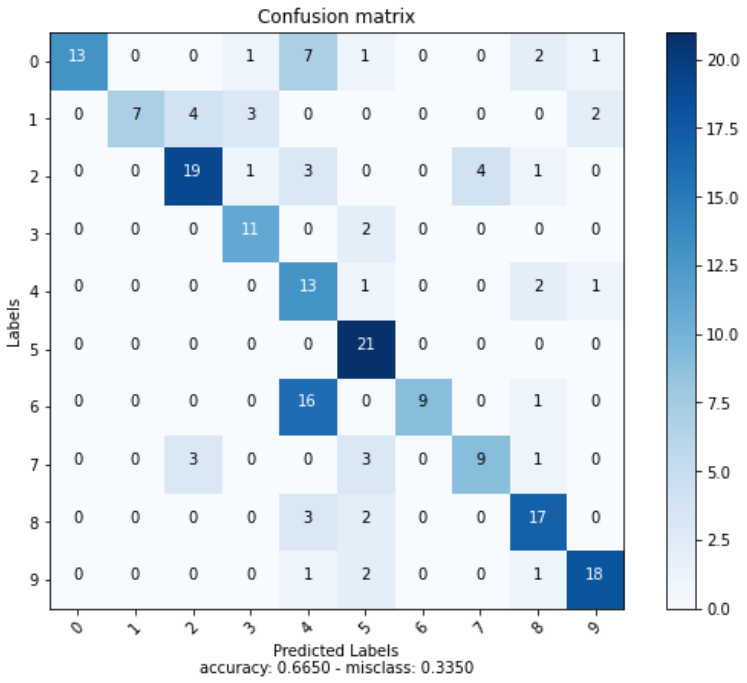
Ο βέλτιστος αριθμός εποχών εκπαίδευσης θα είναι 63-20=43 εποχές. Το early stopping που έχουμε ορίσει έχει patience=20 epochs. Το νευρωνικό σταμάτησε να εκπαιδεύεται στις 63 εποχές, αυτό σημαίνει ότι η $63^{η}$ εποχή ήταν η $20^{η}$ συνεχόμενη εποχή στην οποία το accuracy δεν βελτιώθηκε. Άρα στην $43^{η}$ εποχή το νευρωνικό δίκτυο θα σταματούσε ιδανικά να εκπαιδεύεται. Από την εποχή αυτή και μετά το δίκτυο ξεκινάει να κάνει overfit.

Οι μετρικές που εξήφθηκαν για το βέλτιστο αριθμό εποχών στο test batch παρουσιάζονται παρακάτω:

- Το accuracy είναι (όπως προαναφέρθηκε) 66.5%
  Accuracy είναι ο λόγος του αριθμού των σωστών προβλέψεων προς τον συνολικό αριθμό των προβλέψεων.
- Το precision για κάθε κλάση είναι για τις [0,1,2,3,4,5,6,7,8,9] αντίστοιχα:
  [0.8, 0.52, 1, 1, 0.43, 0.74, 0.51, 0.53, 0.82, 1]
  Το precision είναι ο λόγος των σωστών θετικών προβλέψεων για κάθε κλάση προς

τον συνολικό αριθμό θετικών προβλέψεων για κάθε κλάση ($\frac{true\ positives}{true\ positives+false\ positives}$).

- Το recall για κάθε κλάση είναι για τις [0,1,2,3,4,5,6,7,8,9] αντίστοιχα:
  [0.8, 0.88, 0.25, 0.69, 0.59, 0.95, 0.81, 0.62, 0.41, 0.77]
  Το recall είναι ο λόγος των σωστών θετικών προβλέψεων για κάθε κλάση προς τον αριθμό σωστών θετικών προβλέψεων και λάθος αρνητικών προβλέψεων για κάθε κλάση ($\frac{true\ positives}{true\ positives+false\ negatives}$).
- Το f1 score για κάθε κλάση είναι για τις [0,1,2,3,4,5,6,7,8,9] αντίστοιχα:
  [0.8, 0.65, 0.4, 0.82, 0.5, 0.83, 0.63, 0.57, 0.55, 0.87]
  Το f1 score είναι $2 * \frac{precision*recal}{precision+recall}$. Ουσιαστικά δείχνει την ισορροπία μεταξύ precision και recall.
- Το confusion matrix παρουσιάζεται στην παρακάτω εικόνα:



Από τον confusion matrix οι περισσότερες κλάσεις φαίνεται να διαχωρίζονται αρκετά καλά. Αυτές που δυσκολεύουν περισσότερο το δίκτυο φαίνεται να είναι οι 4 και 6, στις οποίες φαίνεται να μην ξεχωρίζει καλά ο διαχωρισμός. Συγκεκριμένα φαίνεται πως στις εικόνες που απεικονίζουν τον αριθμό ‘6’ το δίκτυο δυσκολεύεται να επιλέξει την κατανομή τους στην κλάση ‘4’ ή ‘6’. Το ίδιο συμβαίνει και σε άλλες κλάσεις (πχ. κλάση ‘1’

σε λίγο μικρότερο βαθμό). Η καλύτερα κατανεμημένη κλάση φαίνεται να είναι η ‘5’ όσον αφορά τα true positives, ενώ η ‘1’ φαίνεται να μην έχει καθόλου false negatives.

Για του λόγου το αληθές τα αποτελέσματα φαίνονται και στην παρακάτω εικόνα:

```
Precision : [0.8        0.51851852 1.         1.         0.43478261 0.74074074
 0.51219512 0.52631579 0.81818182 1.        ]
Recall : [0.8        0.875      0.25       0.69230769 0.58823529 0.95238095
 0.80769231 0.625      0.40909091 0.77272727]
F1_Score : [0.8        0.65116279 0.4        0.81818182 0.5        0.83333333
 0.62686567 0.57142857 0.54545455 0.87179487]
```

c. Δοκιμάστε να κανονικοποιήσετε τις εικόνες στο διάστημα [-1,1]. Ταυτόχρονα, προσθέστε την συνάρτηση ενεργοποίησης (activation function) ReLU (Rectified Linear Unit) ως έξοδο κάθε νευρώνα στα κρυφά επίπεδα (hidden layer). Υπάρχει αλλαγή στην ακρίβεια (accuracy); Ο αριθμός των εποχών πριν συμβεί το overfit έχει μεταβληθεί; Γιατί; Ποια είναι η μεταβολή στον αριθμό των παραμέτρων του δικτύου;

```
min=-1
max=1
```

```
model = tf.keras.Sequential([
  layers.Flatten(),
  Lambda(lambda x: (x-min) / (max-min)),
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.Dense(units=256,activation="relu"),
  layers.Dense(units=128,activation="relu"),
  layers.Dense(units=num_classes)
])
```

Με την προσθήκη των συναρτήσεων ReLU και την κανονικοποίηση των δεδομένων παρατηρούμε ότι γίνεται early stopping νωρίτερα στο μοντέλο (πριν γινόταν 63 εποχές ενώ τώρα μόλις 33), άρα αυτό εκπαιδεύεται γρηγορότερα.

Επίσης, φαίνεται πως υπάρχει αρκετή βελτίωση στην ακρίβεια του δικτύου, η οποία πλέον φτάνει το 75.73% (σε σχέση με την προηγούμενη υλοποίηση που έδινε ακρίβεια μόλις 66.5%).

```
Epoch 00033: accuracy did not improve from 1.00000
46/46 [==============================] - 4s 80ms/step - loss: 0.1123 - accuracy: 0.9605 -
Epoch 00033: early stopping
```

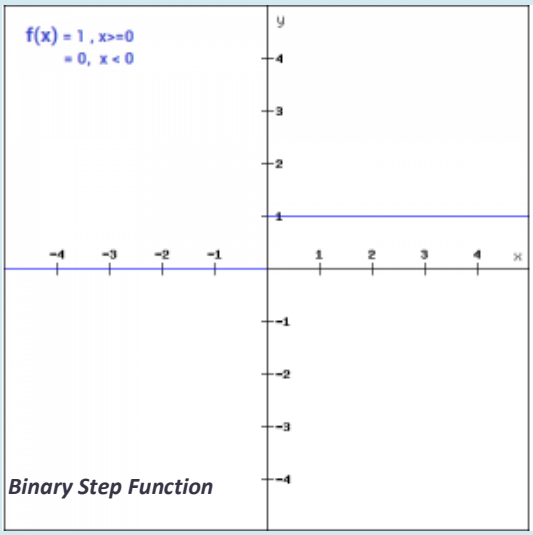
```
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.1,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size,
  shuffle="true",
  color_mode="grayscale")

model.load_weights("weights.best.hdf5")
score = model.evaluate(test_ds)
print('Test score: ', score[0])
print('Test accuracy: ', score[1])
```

```
Found 2062 files belonging to 10 classes.
Using 206 files for validation.
7/7 [==============================] - 0s 6ms/step - loss: 0.8326 - accuracy: 0.7573
Test score:  0.8326131701469421
Test accuracy:  0.7572815418243408
```

Οι παρατηρήσεις αυτές είναι αναμενόμενες. Για να εξηγηθούνε, όμως, θα πρέπει πρώτα να αναλυθεί η λειτουργία των 2 συναρτήσεων ενεργοποίησης (της linear που ήταν η default αρχική συνάρτηση και της relu). Με την ευκαιρία αυτή θα αναλυθούν και άλλες συναρτήσεις ενεργοποίησης, καθώς η γνώση αυτή θα είναι χρήσιμη στο ερώτημα ε.

Στα νευρωνικά δίκτυα η συνάρτηση ενεργοποίησης είναι υπεύθυνη για τη μετατροπή του συνολικού αθροίσματος των βαρών εισόδου στην ενεργοποίηση του νευρώνα ή όχι.

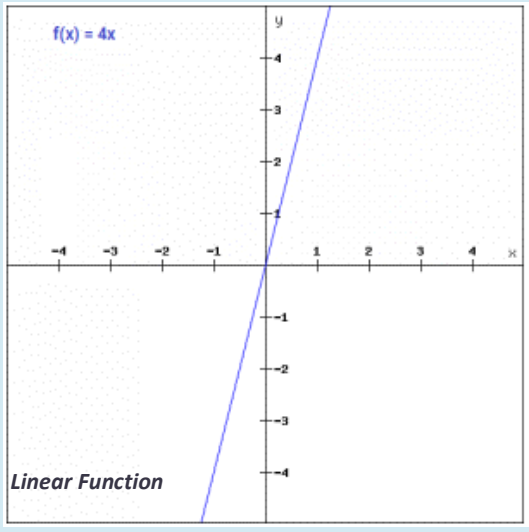
Η **“ Binary Step Function”** είναι ένας threshold based classifier που δίνει ως αποτέλεσμα 0, αν το άθροισμα των βαρών που έχει ως input είναι αρνητικό ή 0, και 1 αν το άθροισμα των βαρών που έχει ως input είναι θετικό.
Ωστόσο, το gradient της συνάρτησης αυτής είναι παντού 0. Αυτό είναι μεγάλο εμπόδιο για το back propagation, δηλαδή του επανασυντονισμού (της βελτίωσης) των βαρών με βάση το ποσοστό του σφάλματος που αποκτήθηκε από την προηγούμενη εποχή. Αυτό συμβαίνει καθώς δεν γνωρίζουμε πόσο ρόλο έπαιξε το κάθε βάρος input του νευρώνα για την λήψη της απόφασης του αποτελέσματος.



Binary Step Function

Η συνάρτηση αυτή έχει καλύτερα αποτελέσματα όταν χρησιμοποιείται για τη δημιουργία ενός binary classifier, δηλαδή το διαχωρισμό των στοιχείων σε 2 διαφορετικές κλάσεις. Δεν είναι τόσο αποδοτική για το διαχωρισμό πολλών διαφορετικών κλάσεων γι’ αυτό και δεν ενδείκνειται για την εκπαίδευση του παρόντος dataset το οποίο περιλαμβάνει 10 κλάσεις συνολικά.

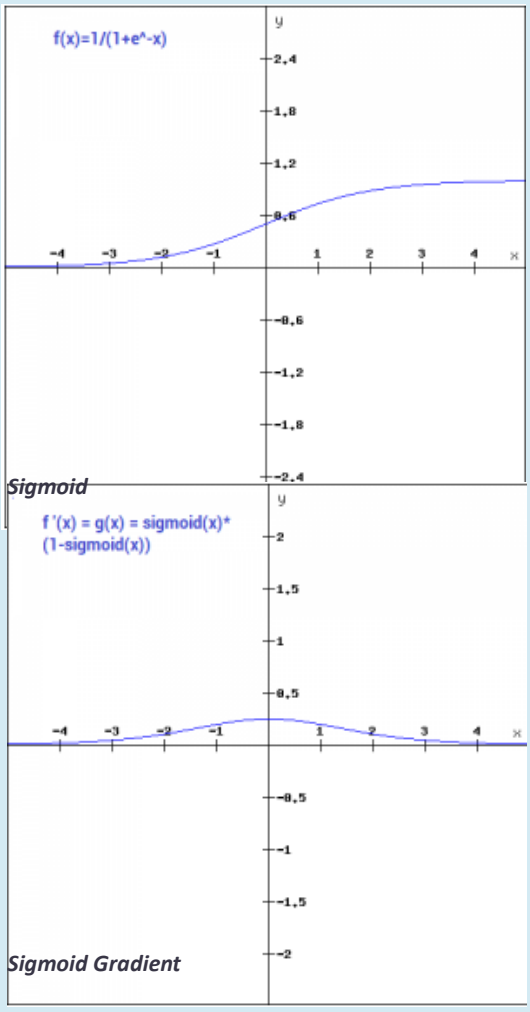
Η **“ Linear Function”** δίνει αποτέλεσμα ανάλογο του αθροίσματος των βαρών του input. Ουσιαστικά δίνει ως αποτέλεσμα f(x)=a*x.
Η συνάρτηση αυτή, παρόλο που το gradient της δεν είναι 0 όπως της προηγούμενης, είναι πάλι σταθερό παντού και δεν εξαρτάται πάλι από το input.
Ως αποτέλεσμα, τα βάρη θα επαναπρογραμματιστούν αλλά ο updating factor θα μένει ίδιος είτε το αποτέλεσμα που έβγαλε ο νευρώνας είναι σωστό είτε λάθος. Το νευρωνικό δίκτυο οπότε δεν θα έχει ουσιαστική βελτίωση σε κάθε εποχή και γι’ αυτό η συνάρτηση αυτή



Linear Function

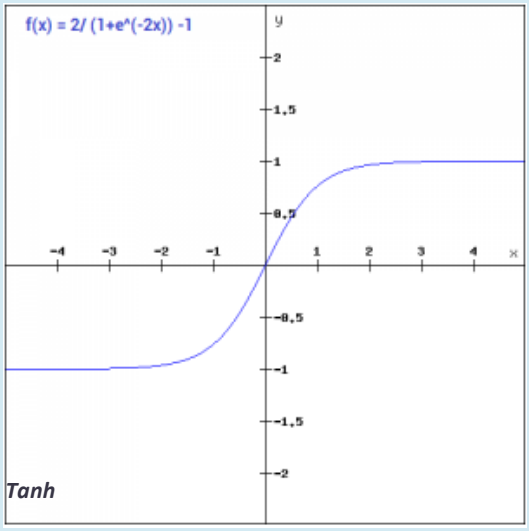
προτιμάται για απλούς διαχωρισμούς και όχι πολύ πολύπλοκους με πολλά χαρακτηριστικά.

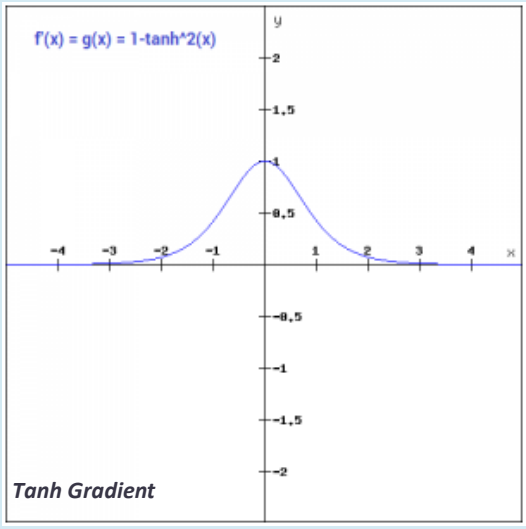
Η **"Sigmoid"** είναι μία πολύ συχνά χρησιμοποιούμενη μη-γραμμική συνάρτηση ενεργοποίησης. Η συνάρτηση αυτή μετατρέπει το άθροισμα των inputs σε μία έξοδο που κυμαίνεται στο διάστημα [0,1] σύμφωνα με τον τύπο $f(x) = \frac{1}{1+e^{-x}}$.
Για την περιγραφή της συνάρτησης αυτής θα πρέπει να παρατηρήσουμε και το gradient της.



Παρατηρούμε ότι οι σημαντικές τιμές του gradient βρίσκονται (και είναι μεταβαλόμενες) μέσα στο διάστημα [-3,3]. Έξω από το διάστημα αυτό οι τιμές της Sigmoid είναι περίπου 0 και ο νευρώνας δεν εκπαιδεύεται ουσιαστικά.
Επιπλέον, άλλο έν καλό χαρακτηριστικό της Sigmoid είναι ότι το output δεν είναι συμμετρικό γύρω από το 0, άρα υπάρχει διαχωρισμός ανάμεσα στους νευρώνες με λάθος αποτέλεσμα και σε αυτούς με σωστό αποτέλεσμα. Το αποτέλεσμα είναι όμως πάντας θετικό.
Η Sigmoid δίνει ως αποοτέλεσμα μία τιμή στο διάστημα [0,1] ανάλογα με το πόσο ανήκει ουσιαστικά το data point σε μία συγκεκριμένη κλάση. Για το λόγ αυτό χρησιμοποιείται κυρίως σε binary classification problems, δηλαδή για την κατηγοριοποίηση των data points σε 2 μόνο κλάσεις. Αυτός είναι και ο λόγος που δεν θεωρείται ιδανική να χρησιμοποιηθεί εδώ.

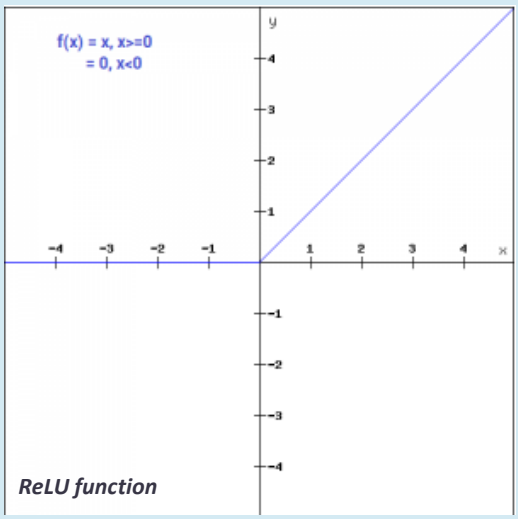
Η **“ Tanh”** μοιάζει πολύ με τη sigmoid, με μόνη διαφορά ότι αυτή είναι συμμετρική γύρω από το 0. Το αποτέλεσμα κυμαίνεται στο διάστημα [-1,-1] με αποτέλεσμα, σε αντίθεση με τη sigmoid το output να μην είναι πάντα του ίδιου προσήμου.

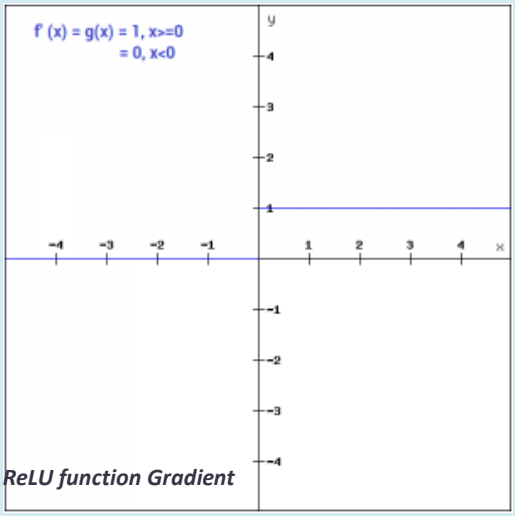




Αν παρατηρήσουμε το αποτέλεσμα του gradient θα δούμε ότι μοιάζει πολύ με της sigmoid απλώς η καμπάνα που σχηματίζει είναι πιο έντονη. Σε σχέση με την sigmoid γενικά προτιμάται η Tanh καθώς είναι zero-centered και τα gradient δεν περιορίζονται να κινούνται μόνο προς μία κατεύθυνση.

Η **“ReLU” (Rectified Linear Unit)** είναι μία μη-γραμμική συνάρτηση η οποία έχει ως έξοδο το ίδιο το άθροισμα των βαρών που έχει ως input, αν αυτό είναι θετικό, ή το 0 αν το input άθροισμα είναι αρνητικό ή 0. Είναι μία default συνάρτηση ενεργοποίησης στα νευρωνικά δίκτυα.
Το κύριο θετικό χαρακτηριστικό της ReLU είναι ότι δεν ενεργοποιεί όλους τους νευρώνες ενός νευρωνικού δικτύου ταυτόχρονα.

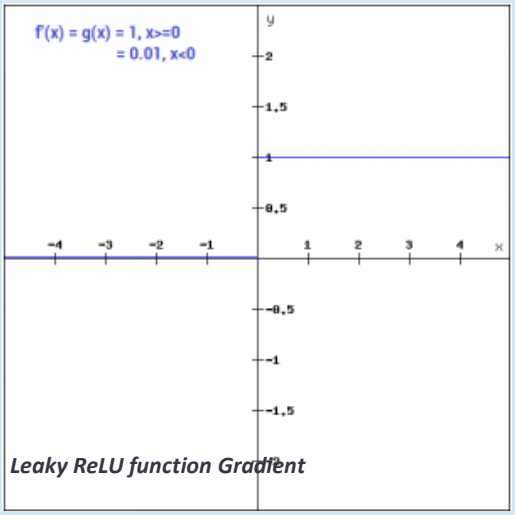




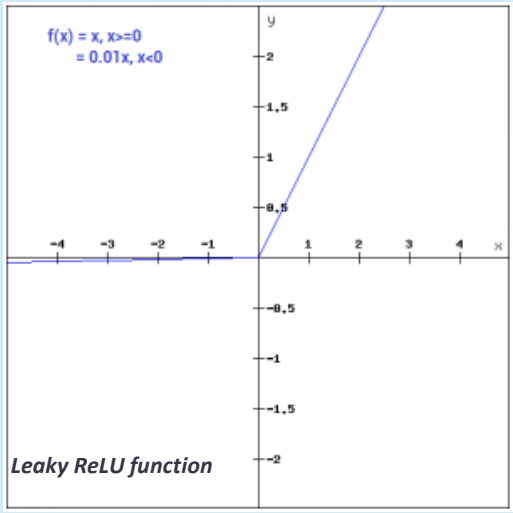
ReLU function Gradient

Παρατηρώντας το gradient της ReLU το γεγονός ότι είναι 0 για τις τιμές που είναι ≤ 0 κατά τη διαδικασία του backpropagation τα βάρη κάποιων νευρώνων δεν επαναπρογραμματίζονται. Αυτό μπορεί να οδηγήσει στη δημιουργία 'νεκρών νευρώνων' που δε χρησιμοποιούνται. Για την αντιμετώπιση της κατάστασης αυτής δημιουργήθηκαν κάποια improved versions της ReLU που παρουσιάζονται στη συνέχεια:

Η **"Leaky ReLU"** είναι η ReLU απλώς αντί για 0 όταν το input άθροισμα του νευρώνα είναι αρνητικό, η έξοδος του νευρώνα είναι 0.01*input.
Το αποτέλεσμα είναι η έξοδος να είναι πάλι πολύ κοντά στο 0 στις τιμές εκείνες (άρα να μη χάνεται τελείως το χαρακτηριστικό της ReLU), ταυτόχρονα όμως το gradient της συνάρτησης στις τιμές εκείνες να μην είναι 0, αλλά μία σταθερή τιμή 0.01.



Leaky ReLU function Gradient



Leaky ReLU function

Η **“Softmax”** είναι ουσιαστικά συνδυασμός πολλών sigmoid functions με σκοπό την κατηγοριοποίηση των data points σε πολλές κλάσεις (όχι μόνο σε 2, όπως επιτυγχάνει η απλή Sigmoid). Το αποτέλεσμά της είναι η πιθανότητα να ανήκει το data point σε κάθε κλάση ξεχωριστά. Η συνάρτηση αυτή θα χρησιμοποιηθεί στο τελευταίο στρώμα του νευρωνικού, καθώς η έξοδός της είναι ακριβώς όσες είναι και οι κλάσεις στις οποίες θέλουμε να ταξινομήσουμε τα data points.

Από τις παραπάνω περιγραφές καταλαβαίνουμε ότι η linear δεν βοηθάει γενικά πολύ στην αλλαγή των βαρών των νευρώνων, καθώς δεν αλλάζει ο updating factor, γι’ αυτό και έχουμε πολύ πιο γρήγορη εκπαίδευση με τη ReLU.
Αντίθετα, η ReLU έχει μηδενικές τιμές για αρνητικό input βαρών. Έχει άρα διαφορετική αντιμετώπιση στους νευρώνες που είναι λάθος και σε αυτούς που είναι σωστοί, με αποτέλεσμα να εκπαιδεύεται και πιο γρήγορα και πιο αποτελεσματικά.

Στην μείωση των εποχών και την αύξηση της ακρίβειας βοήθησε, όμως, και το normalization [-1,1] που εφαρμόστηκε:

Το normalization βοηθάει έτσι ώστε χαρακτηριστικά με μεγαλύτερο scale (πχ το χέρι είναι πιο κοντά/πιο μακριά στην κάμερα ή αν μία εικόνα είναι πιο φωτεινή/σκοτεινή από μία άλλη) να έχουν την ίδια βαρύτητα (ενώ το μεγαλύτερο χέρι περιέχει περισσότερη πληροφορία και θα μετράει κανονικά περισσότερο στην εκπαίδευση των νευρώνων, με το normalization θα έχουν και το μικρό και το μεγάλο χέρι το ίδιο range και άρα θα μετράνε το ίδιο).
Επίσης, πολύ μεγάλες τιμές δίνουν πολύ μεγαλύτερο υπολογιστικό βάρος τόσο στο front, όσο και στο back propagation, με αποτέλεσμα το δίκτυο να είναι πολύ πιο αργό απ’ ότι θα ήταν αν είχε υποστεί πρώτα normalization.

```
Total params: 2,594,442
Trainable params: 2,594,442
Non-trainable params: 0
```

Παράμετροι αρχικού δικτύου

```
Total params: 2,594,442
Trainable params: 2,594,442
Non-trainable params: 0
```

Παράμετροι τελικού δικτύου

Τίποτα δεν άλλαξε στον αριθμό των παραμέτρων του δικτύου. Το μόνο που αλλάξαμε ήταν η συγκεκριμενοποίηση των activation functions (που ουσιαστικά και πριν ήταν συγκεκριμένες, αφού χρησιμοποιούνταν η default linear function), γι' αυτό και ο αριθμός των παραμέτρων δεν άλλαξε.

d. Δοκιμάστε παράλληλα με τις προσθήκες του προηγούμενου ερωτήματος να εφαρμόσετε και batch normalization σε κάθε επίπεδο. Ποια είναι η μεταβολή στο μέγεθος του δικτύου; Αναμένεται να υπάρχει και μεταβολή του χρόνου επεξεργασίας του δικτύου; Τι επιτυγχάνει ο αλγόριθμος batch normalization;

Ο αλγόριθμος batch normalization κανονικοποιεί τις εισόδους κάθε layer για κάθε mini-batch. Με τον τρόπο αυτό σταθεροποιείται η διαδικασία της μάθησης και μας δίνεται η δυνατότητα να αυξήσουμε το batch size ώστε κάθε εποχή να ολοκληρώνεται γρηγορότερα.

Πράγματι, φαίνεται αλλαγή στην απόδοση του δικτύου (προς το καλύτερο) και παρατηρούμε ότι οι εποχές έχουν μειωθεί. Το δίκτυο εκπαιδεύεται πλέον πιο αργά σε κάθε εποχή (καθώς έχουμε επιπλέον επεξεργασία για την κανονικοποίηση) αλλά εκπαιδεύεται και πιο σταθερά και χρειάζονται λιγότερα epochs. Το αποτέλεσμα είναι καλύτερο (79.13% αντί 75.73% που ήταν η ακρίβεια χωρίς το batch normalization) και παρουσιάζεται στην παρακάτω εικόνα:

```
Epoch 00021: early stopping
```

```
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.1,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size,
  shuffle="true",
  color_mode="grayscale")

model.load_weights("weights.best.hdf5")
score = model.evaluate(test_ds)
print('Test score: ', score[0])
print('Test accuracy: ', score[1])
```

```
Found 2062 files belonging to 10 classes.
Using 206 files for validation.
4/4 [==============================] - 0s 10ms/step - loss: 0.9418 - accuracy: 0.7913
Test score:  0.9418219327926636
Test accuracy:  0.791262149810791
```

e. Υλοποιήστε διάφορες δομές τεχνητών νευρωνικών δικτύων και εντοπίστε αυτό που επιλύει το πρόβλημα της ταξινόμησης με μεγαλύτερη ακρίβεια. ΣΗΜΕΙΩΣΗ: Μπορείτε για παράδειγμα να πειραματιστείτε με τον αριθμό των επιπέδων, των νευρώνων ανά επίπεδο, τις συναρτήσεις ενεργοποίησης, τον αλγόριθμο βελτιστοποίησης (optimization algorithm), την προ-επεξεργασία των εικόνων του dataset, μείωση διαστάσεων (dimensionality reduction). Είστε ελεύθερες/ελεύθεροι να υλοποιήσετε ότι θεωρείτε ότι θα βοηθήσει στην αύξηση της ακρίβειας.

Στα 2 κρυφά layers επιλέχθηκε αρχικά η χρήση της ReLU καθώς είναι η πιο συνηθισμένη για τα hidden layers και δουλεύει πολύ καλά για την ταξινόμηση πολλών κλάσεων.

Το ίδιο ακριβώς νευρωνικό δίκτυο υλοποιήθηκε και με τη χρήση της Leaky ReLU για την αποφυγή τυχόν νεκρών νευρώνων. Το δίκτυο σταμάτησε να εκπαιδεύεται στα 32 μόλις epochs.

Το αποτέλεσμα φαίνεται να είναι χειρότερο:

```
score = model.evaluate(val_ds)
print('Test score: ', score[0])
print('Test accuracy: ', score[1])
```

```
13/13 [==============================] - 1s 23ms/step - loss: 23.5591 - accuracy: 0.4223
Test score:  23.559099197387695
Test accuracy:  0.4223301112651825
```

Επομένως κρατήθηκε η απλή ReLU.

Με τη χρήση της sigmoid στο πρώτο κυφό επίπεδο δεν παρατηρήθηκε επίσης κάποια βελτίωση:

```
Found 2062 files belonging to 10 classes.
Using 206 files for validation.
4/4 [==============================] - 0s 10ms/step - loss: 1.0934 - accuracy: 0.7524
Test score:  1.0933678150177002
Test accuracy:  0.7524271607398987
```

Έγινε δοκιμή επίσης να προστεθεί ένα ακόμη επίπεδο πριν το επίπεδο της sigmoid:

```
Found 2062 files belonging to 10 classes.
Using 206 files for validation.
4/4 [==============================] - 0s 8ms/step - loss: 1.2308 - accuracy: 0.7621
Test score:  1.2308284044265747
Test accuracy:  0.762135922908783
```

Πάλι τα αποτελέσματα ήταν απογοητευτικά.

Εφαρμόστηκε η ReLU σε όλα τα κρυφά επίπεδα. Τώρα εφαρμόζεται άλλος optimizer: Ο SGD.

```
Found 2062 files belonging to 10 classes.
Using 206 files for validation.
4/4 [==============================] - 0s 19ms/step - loss: 0.9436 - accuracy: 0.6990
Test score:  0.9436301589012146
Test accuracy:  0.6990291476249695
```

Εφαρμόστηκαν επίσης οι RMSprop, Adadelta, Adagrad, Adamax, Nadam, Ftrl, καθώς και να γίνει monitor το validation accuracy αντί το validation loss. Το καλύτερο αποτέλεσμα που προέκυψε ήταν:

```
Found 2062 files belonging to 10 classes.
Using 206 files for validation.
2/2 [==============================] - 0s 40ms/step - loss: 81.4512 - accuracy: 0.8155
Test score:  81.45117950439453
Test accuracy:  0.8155339956283569
```

Με τη χρήση του adam.

Δοκιμάστηκε να γίνει horizontal flip, width και height shift, rotation, shear και zoom:

```
tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=5, width_shift_range=0.1,
    height_shift_range=0.1, shear_range=0.1, zoom_range=0.2,
    horizontal_flip=True, vertical_flip=False
)
```

```
Found 2062 files belonging to 10 classes.
Using 206 files for validation.
2/2 [==============================] - 0s 43ms/step - loss: 100.0663 - accuracy: 0.7718
Test score:  100.06632995605469
Test accuracy:  0.7718446850776672
```

Το αποτέλεσμα χειροτέρευε.

Χωρίς το horizontal flip που βλέποντας τις εικόνες παρατηρούμε ότι δεν χρειάζεται καθώς όλα τα νούμερα παριστάνονται με το ίδιο χέρι (αριστερό):

```
Found 2062 files belonging to 10 classes.
Using 206 files for validation.
2/2 [==============================] - 0s 38ms/step - loss: 100.3995 - accuracy: 0.8155
Test score:  100.39952850341797
Test accuracy:  0.8155339956283569
```

Το αποτέλεσμα γίνεται καλύτερο.
(Στο .ipynb αρχείο θα φανεί απόδοση `0.7804877758026123`. Αυτό συμβαίνει γιατί κάθε φορά που ξανατρέχω τον κώδικα γίνονται shuffle οι εικόνες και δημιουργούνται διαφορετικά datasets που συνεπώς δίνουν και μικρές διαφορές στην απόδοση. Εγώ επέλεξα να σας δείξω και την καλύτερη που βρήκα).

Καταλήγουμε στη χρήση του παρακάτω δικτύου ως βέλτιστου:

```
Model: "sequential_42"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_43 (Flatten)         (1, 10000)                0
_________________________________________________________________
lambda_44 (Lambda)           (1, 10000)                0
_________________________________________________________________
rescaling_44 (Rescaling)     (1, 10000)                0
_________________________________________________________________
dense_141 (Dense)            (1, 1024)                 10241024
_________________________________________________________________
dense_142 (Dense)            (1, 512)                  524800
_________________________________________________________________
dense_143 (Dense)            (1, 256)                  131328
_________________________________________________________________
dense_144 (Dense)            (1, 10)                   2570
=================================================================
Total params: 10,899,722
Trainable params: 10,899,722
Non-trainable params: 0
_________________________________________________________________
```

Συγκεκριμένα, το δίκτυο αποτελείται από ένα flatten επίπεδο, ένα επίπεδο κανονικοποίησης στο διάστημα [-1,1], ένα rescaling επίπεδο, dropout 0.1, 3 επίπεδα 1024, 512 και 256 νευρώνων αντίστοιχα, με συνάρτηση ενεργοποίησης και των 3 τη ReLU, και ένα επίπεδο εξόδου 10 νευρώνων με συνάρτηση ενεργοποίησης την softmax. Το batch size τέθηκε 150. Με μεγαλύτερο batch size υπάρχει μεγαλύτερη ποικιλία εικόνων σε κάθε επανάληψη της εποχής και το δίκτυο εκπαιδεύεται πιο γενικά. Monitor για το καλύτερο checkpoint γίνεται το validation accuracy και για το early stopping γίνεται monitored το accuracy του train.

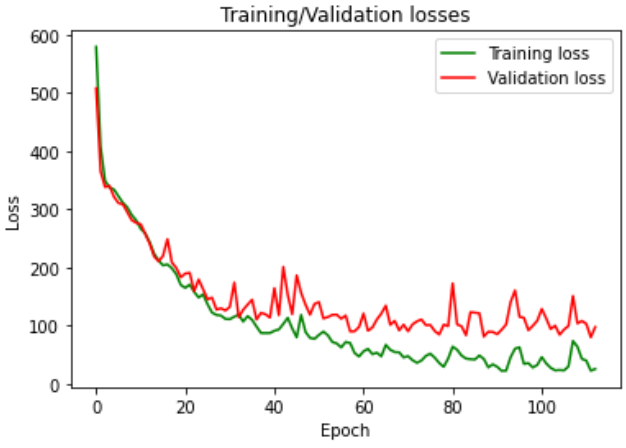
```
model = tf.keras.Sequential([
  layers.Flatten(),
  Lambda(lambda x: (x-min) / (max-min)),
  layers.experimental.preprocessing.Rescaling(1./255),
  layer.Dropout(0.1),
  layers.Dense(units=1024,activation="relu"),
  layers.Dense(units=512,activation="relu"),
  layers.Dense(units=256,activation="relu"),
  layers.Dense(units=num_classes,activation="softmax")
])
```

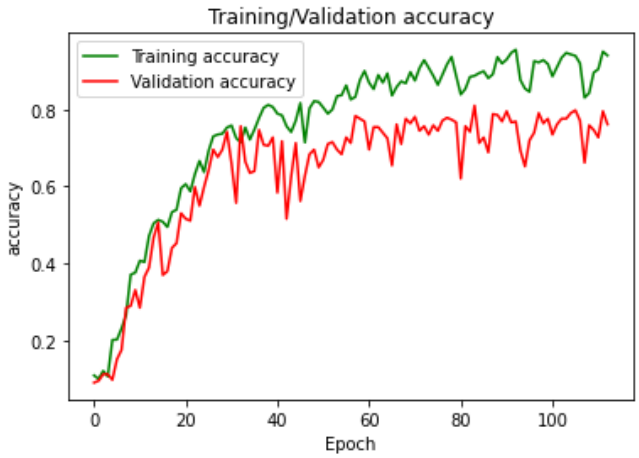
```
tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=10, width_shift_range=0.1,
    height_shift_range=0.1, shear_range=0.1, zoom_range=0.2
)
```

Κάνει, επίσης, περιστροφή 10 μοιρών, 0.1 width και height shift καθώς και shear range και 0.2 zoom range.

```
Found 2062 files belonging to 10 classes.
Using 206 files for validation.
2/2 [==============================] - 0s 32ms/step - loss: 0.8669 - accuracy: 0.8301
Test score:  0.8668835759162903
Test accuracy:  0.8300970792770386
```

Το βέλτιστο αποτέλεσμα είναι 83.01%.





a. Ξεκινήστε χρησιμοποιώντας δυο επίπεδα συνέλιξης με 64 και 32 φίλτρα αντίστοιχα με μέγεθος παραθύρου 3x3, και ένα fully connected επίπεδο με 128 νευρώνες. Μετά από κάθε συνελικτικό επίπεδο χρησιμοποιήστε επίπεδο υποδειγματοληψίας (pooling), προτείνεται max pooling.

Χρησιμοποιήθηκε η Maxpool, όπως προτείνεται. Το δίκτυο που υλοποιήθηκε σύμφωνα με τις υποδείξεις είναι το εξής:

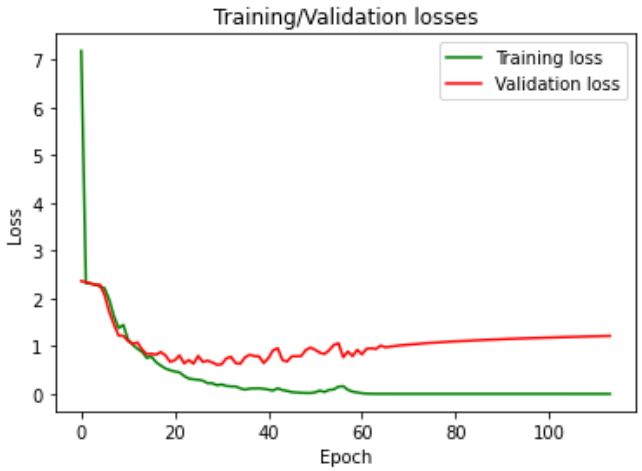
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 98, 98, 64)        640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 47, 47, 32)        18464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 16928)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               2166912
=================================================================
Total params: 2,186,016
Trainable params: 2,186,016
Non-trainable params: 0
_________________________________________________________________
```

Όπως φαίνεται έχουμε ένα convolution layer 64 φίλτρων και 3x3 παράθυρο. Ακολουθεί ένα επίπεδο υποδειγματοληψίας παραθύρου 2x2, άλλο ένα convolution layer 32 φίλτρων και 3x3 παράθυρο και άλλο ένα επίπεδο υποδειγματοληψίας παραθύρου 2x2. Στη συνέχεια γίνεται flatten ώστε να επιπεδοποιηθούν όλα τα χαρακτηριστικά που έχουν συλλεχθεί και ακολουθεί το fully connected κομμάτι του

δικτύου όπου θα γίνει το classification. Το κομμάτι αυτό δεν είναι παρά ένα Layer 128 νευρώνων με (προς το παρόν) την default συνάρτηση ενεργοποίησης την 'linear'.
Το αποτέλεσμα είναι ενθαρρυντικό, καθώς ήδη επιτεύχθηκε 76.7% ακρίβεια με πολύ απλές σχετικά ρυθμίσεις:

```
Found 2062 files belonging to 10 classes.
Using 206 files for validation.
4/4 [==============================] - 2s 325ms/step - loss: 2.1133 - accuracy: 0.7670
Test score:  2.1133251190185547
Test accuracy:  0.7669903039932251
```

b. Παραγωγή των γραφημάτων train/test – loss ανά εποχή. Εμφανίζεται το πρόβλημα underfitting/overfitting; Αν ναι, να προτείνετε έναν τρόπο διαχείρισης του φαινομένου. Επίσης να αιτιολογήσετε ποιος είναι ο βέλτιστος αριθμός εποχών εκπαίδευσης. Για τον βέλτιστο αριθμό εποχών εκπαίδευσης, ζητείται να εξαχθούν οι μετρικές: accuracy, precision recall, f1 score, confusion matrix.



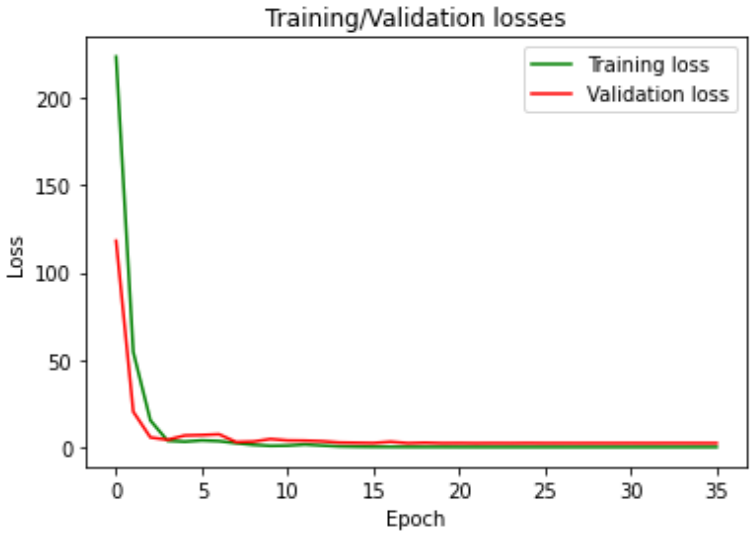
Παρατηρείται αρχικά (ακόμη πιο ξεκάθαρα από ότι παρατηρήθηκε στο ANN δίκτυο) overfitting, στη συνέχεια όμως εφαρμόζεται η τεχνική που προαναφέρθηκε στο ANN, στην οποία βάζουμε early stopping και checkpoint για να περιορίσουμε ακριβώς αυτό το φαινόμενο. Όπως φαίνεται στην επόμενη εικόνα το overfit δεν

συμβαίνει πλέον και το αποτέλεσμα είναι πλέον πολύ καλύτερο (76.7%):

```
Epoch 00036: early stopping
```

```
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.1,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size,
  shuffle="true",
  color_mode="grayscale")

model.load_weights("weights.best.hdf5")
score = model.evaluate(test_ds)
print('Test score: ', score[0])
print('Test accuracy: ', score[1])
```

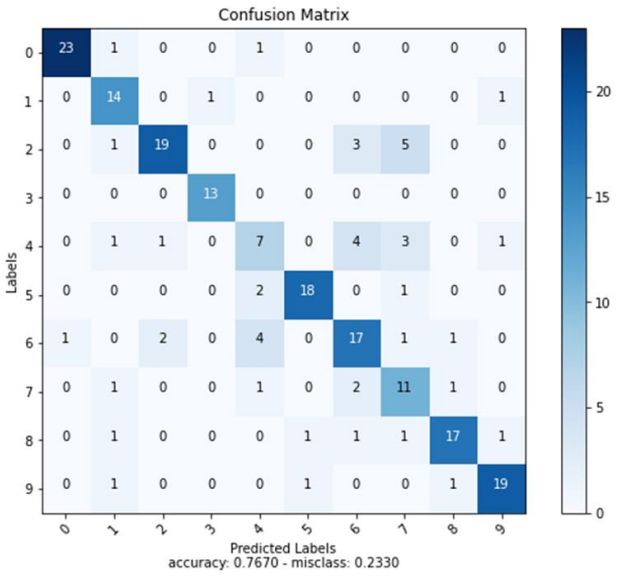
```
Found 2062 files belonging to 10 classes.
Using 206 files for validation.
4/4 [==============================] - 2s 325ms/step - loss: 2.1133 - accuracy: 0.7670
Test score:  2.1133251190185547
Test accuracy:  0.7669903039932251
```



Ο βέλτιστος αριθμός εποχών εκπαίδευσης είναι 16 (36 – 20 epochs = 16). Από το διπλανό σχήμα φαίνεται ο σημαντικός περιορισμός του overfitting.

Οι μετρικές που εξήφθηκαν για το βέλτιστο αριθμό εποχών στο test batch παρουσιάζονται παρακάτω:

- Το accuracy είναι (όπως προαναφέρθηκε) 76.7%
  Accuracy είναι ο λόγος του αριθμού των σωστών προβλέψεων προς τον συνολικό αριθμό των προβλέψεων.
- Το precision για κάθε κλάση είναι για τις [0,1,2,3,4,5,6,7,8,9] αντίστοιχα:
  [0.96, 0.7, 0.86, 0.93, 0.47, 0.9, 0.63, 0.5, 0.85, 0.86].
  Το precision είναι ο λόγος των σωστών θετικών προβλέψεων για κάθε κλάση προς τον συνολικό αριθμό θετικών προβλέψεων για κάθε κλάση ($\frac{true\ positives}{true\ positives+false\ positives}$).
- Το recall για κάθε κλάση είναι για τις [0,1,2,3,4,5,6,7,8,9] αντίστοιχα:
  [0.92, 0.88, 0.68, 0.41, 0.86, 0.0.65, 0.69, 0.77, 0.86].
  Το recall είναι ο λόγος των σωστών θετικών προβλέψεων για κάθε κλάση προς τον αριθμό σωστών θετικών προβλέψεων και λάθος αρνητικών προβλέψεων για κάθε κλάση ($\frac{true\ positives}{true\ positives+false\ negatives}$).
- Το f1 score για κάθε κλάση είναι για τις [0,1,2,3,4,5,6,7,8,9] αντίστοιχα:
  [0.94, 0.78, 0.76, 0.96, 0.44, 0.88, 0.64, 0.58, 0.81, 0.86].
  Το f1 score είναι $2 * \frac{precision*recal}{precision+recall}$. Ουσιαστικά δείχνει την ισορροπία μεταξύ precision και recall.
- Το confusion matrix παρουσιάζεται στην παρακάτω εικόνα:



Από τον confusion matrix οι περισσότερες κλάσεις φαίνεται να διαχωρίζονται αρκετά καλά. Αυτές που δυσκολεύουν περισσότερο το δίκτυο φαίνεται να είναι οι 4 και 7, στις οποίες φαίνεται να μην ξεχωρίζει καλά ο διαχωρισμός.

Για του λόγου το αληθές τα αποτελέσματα φαίνονται και στην παρακάτω εικόνα:

```
Precision : [0.95833333 0.7        0.86363636 0.92857143 0.46666667 0.9
 0.62962963 0.5        0.85       0.86363636]
Recall : [0.92       0.875      0.67857143 1.         0.41176471 0.85714286
 0.65384615 0.6875     0.77272727 0.86363636]
F1_Score : [0.93877551 0.77777778 0.76       0.96296296 0.4375     0.87804878
 0.64150943 0.57894737 0.80952381 0.86363636]
Samples per class : [25 16 28 13 17 21 26 16 22 22]
```

c. Υλοποιήστε διάφορες δομές συνελικτικών τεχνητών νευρωνικών δικτύων και μεταβλητού αριθμού νευρώνων.

Δοκιμάστηκε να αλλάξει ο αριθμός των νευρώνων στη Dense, ο αριθμός των φίλτρων στα convolutional layers, τα pool sizes και να προστεθούν επιπλέον επίπεδα (τόσο στο convolutional όσο και στο fully connected layer).

Ο μέγιστος αριθμός εποχών που δόθηκαν για την εκπαίδευση του δικτύου αυξήθηκε σε 500, καθώς παρατηρήθηκε ότι το CNN χρειάζεται περισσότερα epochs για να εκπαιδευτεί αποτελεσματικά.

Το καλύτερο αποτέλεσμα που επιτεύχθηκε ήταν το παρακάτω:

```
num_classes = 10

min=-1
max=1

filepath = "weights.best.hdf5"
checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1,
                             save_best_only=True, save_weights_only=True,
                             mode='auto', save_freq='epoch')

early = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=50,
                      verbose=1, mode='auto')
```

```
model = tf.keras.Sequential([
  layers.Conv2D(64,(1,1),activation="relu"),
  layers.Conv2D(128,(3,3),activation="relu"),
  layers.MaxPooling2D(pool_size=(4,4)),
  layers.Conv2D(256,(1,1),activation="relu"),
  layers.Conv2D(512,(1,1),activation="relu"),
  layers.Conv2D(1024,(3,3),activation="relu"),
  layers.MaxPooling2D(pool_size=(6,6)),

  layers.Flatten(),
  layers.Dense(1024, activation='relu'),
  layers.Dense(512, activation='relu'),
  layers.Dense(256, activation='relu'),
  layers.Dense(128, activation='relu'),
  layers.Dense(64, activation='relu'),
  layers.Dense(32, activation='relu'),
  layers.Dense(num_classes, activation='softmax')
])
```

```
[30] model.compile(
       optimizer='adam',
       loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
       metrics=['accuracy'])
```

```
Found 2062 files belonging to 10 classes.
Using 206 files for validation.
2/2 [==============================] - 1s 622ms/step - loss: 0.9885 - accuracy: 0.8738
Test score:  0.9884524941444397
Test accuracy:  0.8737863898277283
```

Δηλαδή, το καλύτερο μοντέλο που επιτεύχθηκε είχε accuracy 87.38%.

d. CNN tuning: Χρησιμοποιήστε κανονικοποίηση των δεδομένων σας (batch normalization). Τι επίδραση έχει στην ακρίβεια;

```
model = tf.keras.Sequential([
  layers.BatchNormalization(),
  layers.Conv2D(64,(1,1),activation="relu"),
  layers.Conv2D(128,(3,3),activation="relu"),
  layers.MaxPooling2D(pool_size=(4,4)),
  layers.BatchNormalization(),
  layers.Conv2D(256,(1,1),activation="relu"),
  layers.BatchNormalization(),
  layers.Conv2D(512,(1,1),activation="relu"),
  layers.Conv2D(1024,(3,3),activation="relu"),
  layers.BatchNormalization(),
  layers.MaxPooling2D(pool_size=(6,6)),

  layers.Flatten(),
  layers.Dense(1024, activation='relu'),
  layers.BatchNormalization(),
  layers.Dense(512, activation='relu'),
  layers.BatchNormalization(),
  layers.Dense(256, activation='relu'),
  layers.BatchNormalization(),
  layers.Dense(128, activation='relu'),
  layers.BatchNormalization(),
  layers.Dense(64, activation='relu'),
  layers.BatchNormalization(),
  layers.Dense(32, activation='relu'),
  layers.BatchNormalization(),
  layers.Dense(num_classes, activation='softmax')
])
```

Batch normalization έγινε πριν από κάθε convolutional και fully connected layer.

Δε θα είχε νόημα να μπει πριν από max pooling ή πριν από το flatten, καθώς τα συγκεκριμένα layers δεν περιέχουν νευρώνες, απλώς επεξεργάζονται τα δεδομένα με σκοπό να εκπαιδευτούν οι νευρώνες από το επόμενο επίπεδο καλύτερα.

Τα αποτελέσματα που προέκυψαν με τη χρήση του batch normalization φαίνονται παρακάτω:

```
Epoch 00164: early stopping
```

```
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.1,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size,
  shuffle="true",
  color_mode="grayscale")

model.load_weights("weights.best.hdf5")
score = model.evaluate(test_ds)
print('Test score: ', score[0])
print('Test accuracy: ', score[1])
```

```
Found 2062 files belonging to 10 classes.
Using 206 files for validation.
2/2 [==============================] - 0s 58ms/step - loss: 0.1024 - accuracy: 0.9854
Test score:  0.10237021744251251
Test accuracy:  0.9854369163513184
```

Με την εφαρμογή preprocessing image generators δεν φαίνεται να έχουμε καμία βελτίωση:

```
Found 2062 files belonging to 10 classes.
Using 206 files for validation.
2/2 [==============================] - 0s 61ms/step - loss: 0.0795 - accuracy: 0.9854
Test score:  0.07953103631734848
Test accuracy:  0.9854369163513184
```

γι’ αυτό και δε συμπεριλαμβάνονται στο τελικό CNN δίκτυο.

Όπως διαπιστώνεται, η απόδοση του δικτύου βελτιώθηκε και έφτασε πλέον το 98.54%. Συγκριτικά και με την καλύτερη απόδοση του ANN η απόδοση αυτού του δικτύου είναι μακράν καλύτερη.
(Στο .ipynb αρχείο θα φανεί απόδοση `0.9804878234863281`. Αυτό συμβαίνει γιατί κάθε φορά που ξανατρέχω τον κώδικα γίνονται shuffle οι εικόνες και δημιουργούνται διαφορετικά datasets που συνεπώς δίνουν και μικρές διαφορές στην απόδοση. Εγώ επέλεξα να σας δείξω και την καλύτερη που βρήκα)

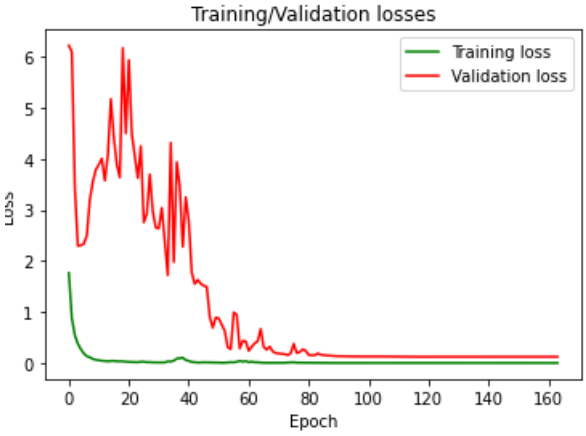
Αυτό είναι και το CNN με τη βέλτιστη ακρίβεια που επιτεύχθηκε.
Αποτελείται από:
batch normalization layer,
convolutional layer των 64 νευρώνων με 1x1 παράθυρο και συνάρτηση ενεργοποίησης τη ‘relu’,
convolutional layer των 128 νευρώνων με 3x3 παράθυρο και συνάρτηση ενεργοποίησης τη ‘relu’,
maxpooling layer με παράθυρο 4x4,
batch normalization layer,
convolutional layer των 256 νευρώνων με 1x1 παράθυρο και συνάρτηση ενεργοποίησης τη ‘relu’,
batch normalization layer,
convolutional layer των 512 νευρώνων με 1x1 παράθυρο και συνάρτηση ενεργοποίησης τη ‘relu’,
convolutional layer των 1024 νευρώνων με 3x3 παράθυρο και συνάρτηση ενεργοποίησης τη ‘relu’,
batch normalization layer,
maxpooling layer με παράθυρο 6x6,

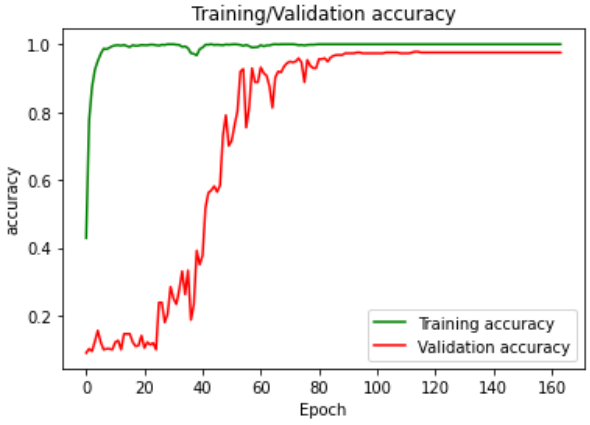
flatten layer,

fully connected layer 1024 νευρώνων και με συνάρτηση ενεργοποίησης τη ‘relu’,

batch normalization layer,
fully connected layer 512 νευρώνων και με συνάρτηση ενεργοποίησης τη ‘relu’,
batch normalization layer,
fully connected layer 256 νευρώνων και με συνάρτηση ενεργοποίησης τη ‘relu’,
batch normalization layer,
fully connected layer 128 νευρώνων και με συνάρτηση ενεργοποίησης τη ‘relu’,
batch normalization layer,
fully connected layer 64 νευρώνων και με συνάρτηση ενεργοποίησης τη ‘relu’,
batch normalization layer,
fully connected layer 32 νευρώνων και με συνάρτηση ενεργοποίησης τη ‘relu’,
batch normalization layer,
fully connected layer 10 νευρώνων και με συνάρτηση ενεργοποίησης τη ‘softmax’.

Optimizer επιλέχθηκε ο ‘adam’ και monitor και στα checkpoints και στο early stopping είναι το validation accuracy.





Model: "sequential_1"

Layer (type)	Output Shape	Param #
batch_normalization_10 (Batc	(None, 100, 100, 1)	4
conv2d_5 (Conv2D)	(None, 100, 100, 64)	128
conv2d_6 (Conv2D)	(None, 98, 98, 128)	73856
max_pooling2d_2 (MaxPooling2	(None, 24, 24, 128)	0
batch_normalization_11 (Batc	(None, 24, 24, 128)	512
conv2d_7 (Conv2D)	(None, 24, 24, 256)	33024
batch_normalization_12 (Batc	(None, 24, 24, 256)	1024
conv2d_8 (Conv2D)	(None, 24, 24, 512)	131584
conv2d_9 (Conv2D)	(None, 22, 22, 1024)	4719616
batch_normalization_13 (Batc	(None, 22, 22, 1024)	4096
max_pooling2d_3 (MaxPooling2	(None, 3, 3, 1024)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_7 (Dense)	(None, 1024)	9438208
batch_normalization_14 (Batc	(None, 1024)	4096
dense_8 (Dense)	(None, 512)	524800
batch_normalization_15 (Batc	(None, 512)	2048
dense_9 (Dense)	(None, 256)	131328
batch_normalization_16 (Batc	(None, 256)	1024
dense_10 (Dense)	(None, 128)	32896
batch_normalization_17 (Batc	(None, 128)	512
dense_11 (Dense)	(None, 64)	8256
batch_normalization_18 (Batc	(None, 64)	256

```
dense_12 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_19 (Batc (None, 32)                128
_________________________________________________________________
dense_13 (Dense)             (None, 10)                330
=================================================================
Total params: 15,109,806
Trainable params: 15,102,956
Non-trainable params: 6,850
```

Συγκριτικά με τον αριθμό των παραμέτρων του ANN δικτύου παρατηρούμε ότι οι παράμετροι του CNN είναι πολύ περισσότερες. Αυτό είναι φυσικό, καθώς τα layers που χρησιμοποιούνται είναι πολύ περισσότερα στο CNN από ότι στο ANN.

```
Total params: 10,899,722
Trainable params: 10,899,722
Non-trainable params: 0
```

e. Δοκιμάστε αρχιτεκτονική CNN χωρίς fully connected layer. Μπορεί να δουλέψει; Ακολούθως χρησιμοποιείστε συνελικτικά επίπεδα χωρίς ενδιάμεσα επίπεδα υποδειγματοληψίας (pooling).

Παραλείποντας το fully connected layer δεν γίνεται ουσιαστικά ταξινόμηση γιατί ακριβώς το fully connected layer είναι αυτό που συνδέει όλους τους νευρώνες μεταξύ τους και με βάση όλα τα χαρακτηριστικά που εξάχθηκαν αποφασίζει σε ποια κατηγορία θα ταξινομηθεί κάθε data point.

Μπορούμε όμως να βάλουμε ως τελευταίο layer ένα convolutional το οποίο θα εφαρμόζει 10 φίλτρα. Έτσι θα μπορούσε να γίνει διαχωρισμός των εικόνων σε 10 κλάσεις. Παρατηρούμε, όμως, ότι κάτι τέτοιο δεν είναι καθόλου αποδοτικό:

```
Test score:  0.4470515549182892
Test accuracy:  0.1061704233288765
```

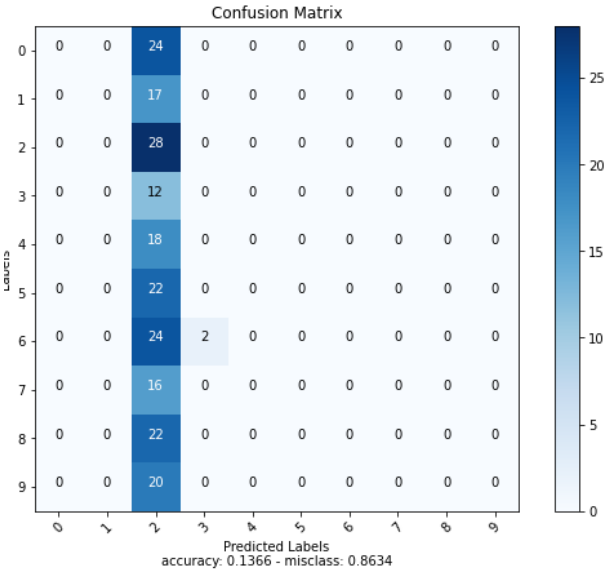
Όπως φαίνεται ουσιαστικά δε γίνεται ταξινόμηση.

Βγάζοντας επιπλέον και τα maxpooling επίπεδα η ακρίβεια πέφτει ακόμη περισσότερο.

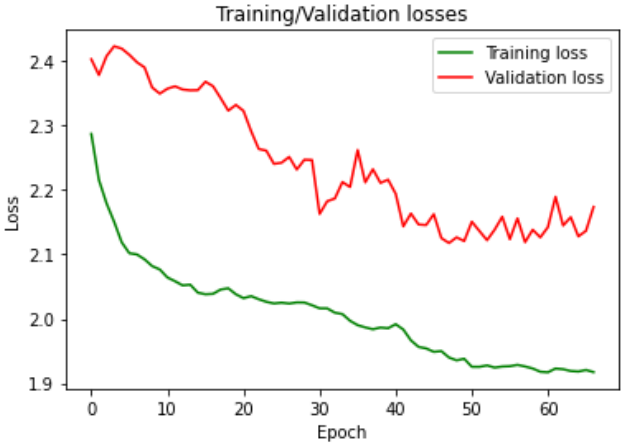
```
model = tf.keras.Sequential([
  layers.BatchNormalization(),
  layers.Conv2D(64,(1,1),activation="relu"),
  layers.Conv2D(128,(3,3),activation="relu"),
  # layers.MaxPooling2D(pool_size=(4,4)),
  layers.BatchNormalization(),
  layers.Conv2D(256,(1,1),activation="relu"),
  layers.BatchNormalization(),
  layers.Conv2D(512,(1,1),activation="relu"),
  layers.Conv2D(1024,(3,3),activation="relu"),
  layers.BatchNormalization(),
  # layers.MaxPooling2D(pool_size=(6,6)),
  layers.Conv2D(10,(3,3),activation="softmax")
])
```

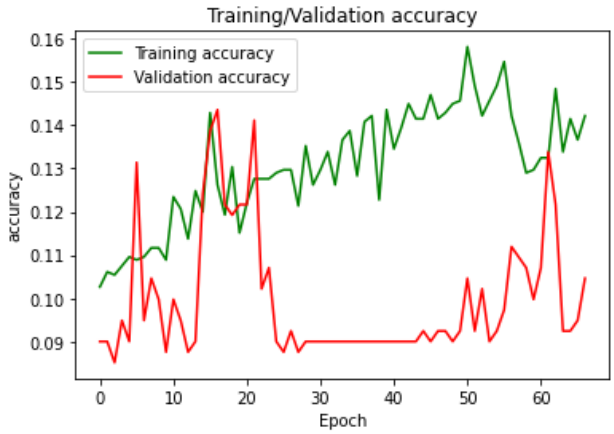
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization_8 (Batch (None, 100, 100, 1)       4
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 100, 100, 64)      128
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 98, 98, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 24, 24, 128)       0
_________________________________________________________________
batch_normalization_9 (Batch (None, 24, 24, 128)       512
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 24, 24, 256)       33024
_________________________________________________________________
batch_normalization_10 (Batc (None, 24, 24, 256)       1024
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 24, 24, 512)       131584
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 22, 22, 1024)      4719616
_________________________________________________________________
batch_normalization_11 (Batc (None, 22, 22, 1024)      4096
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 22, 22, 10)        10250
_________________________________________________________________
global_max_pooling2d (Global (None, 10)                0
=================================================================
Total params: 4,974,094
Trainable params: 4,971,276
Non-trainable params: 2,818
_________________________________________________________________
```

Το δίκτυο που υλοποιήθηκε ήταν ακριβώς το ίδιο με το προηγούμενο, με τη διαφορά ότι τώρα το τελευταίο layer είναι αντί για fully connected ένα convolutional 10 φίλτρων με συνάρτηση ενεργοποίησης softmax και ένα global max pooling.



Η ταξινόμηση φαίνεται πως είναι πολύ κακή.





Βγάζοντας τα maxpooling το δίκτυο σταματάει να μπορεί να εκπαιδευτεί:

Ερωτήσεις Κατανόησης

a. Για ποιον λόγο τα CNN έχουν μεγαλύτερη ακρίβεια;

Το γεγονός ότι τα CNN δίνουν καλύτερη ακρίβεια από τα ANN οφείλεται αρχικά στο γεγονός ότι περιέχουν επίπεδα «εξόρυξης χαρακτηριστικών». Τα επίπεδα αυτά ουσιαστικά αναγνωρίζουν τμήματα της εικόνας, όπως πχ τον κύκλο που σχηματίζεται από το χέρι στον αριθμό ‘0’, και ψάχνει να βρει αυτά στις εικόνες.
Κάθε χαρακτηριστικό επεξεργάζεται από το νευρωνικό δίκτυο ξεχωριστά και ανεξάρτητα από τα υπόλοιπα. Αυτός είναι και ο λόγος που το δίκτυο αυτό δουλεύει πολύ καλύτερα.

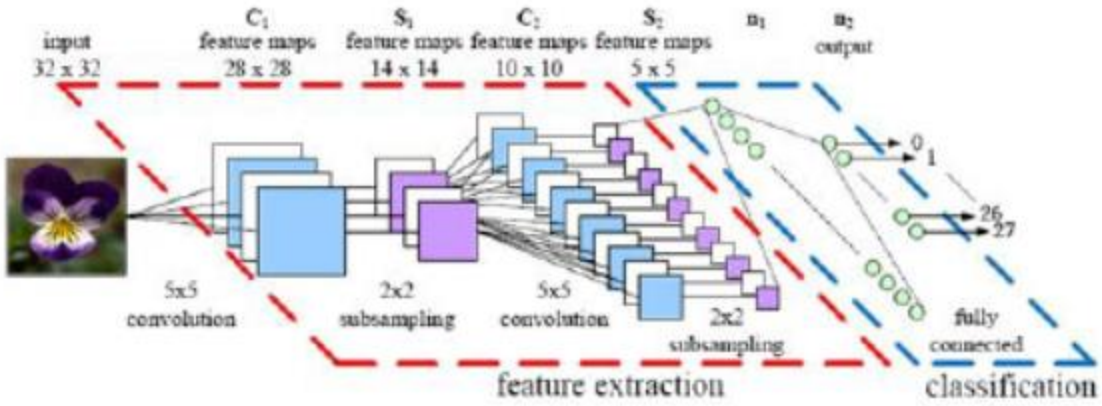


Fig 3. A Convolutional Neural Network (CNN) that shows three types of layers with different hyperparameters

b. Ποια είναι η ειδοποιός διαφορά των CNN και των ANN ως προς την ταξινόμηση εικόνων;

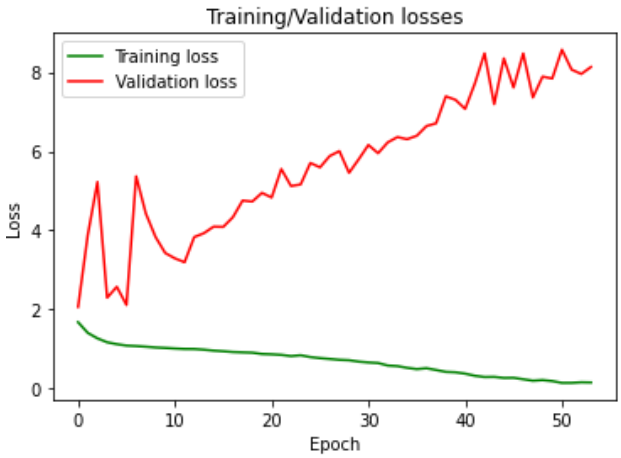
Όπως προαναφέρθηκε στο ερώτημα α. τα CNN περιέχουν επίπεδα «εξόρυξης χαρακτηριστικών». Αυτό τους επιτρέπει να ψάχνουν μικρότερα τμήματα της εικόνας και να αναγνωρίζουν το ίδιο χαρακτηριστικό σε εικόνες της ίδιας κατηγορίας. Αυτό

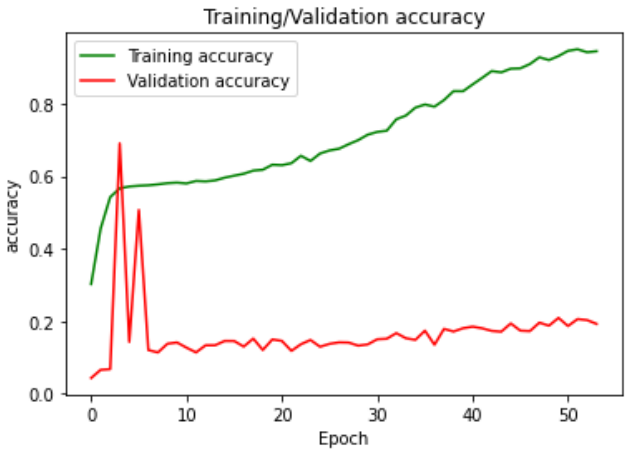
τους δίνει πολύ μεγαλύτερη ακρίβεια, όπως φαίνεται και από τα αποτελέσματα της ταξινόμησης των εικόνων της παρούσας εργασίας.
Η κύρια διαφορά των 2 ειδών δικτύων είναι ότι στα CNN μόνο το τελευταίο layer είναι fully connected, ενώ στα ANN κάθε νευρώνας συνδέεται με όλους τους υπόλοιπους.

BONUS

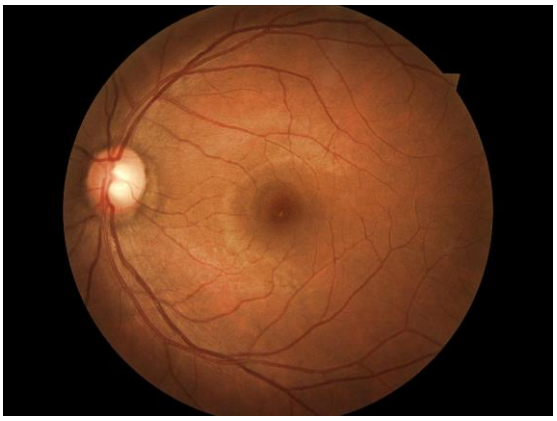
a. Έχουν παρατεθεί και 3 επιπλέον dataset. Επιλέξτε ένα από αυτά και εκτελέστε το πείραμα με την βέλτιστη αρχιτεκτονική CNN που προτείνατε για το αρχικό dataset. Ταυτόχρονα υλοποιείστε την αρχιτεκτονική ResNet-50.

Εκτελώντας το πείραμα με την προηγούμενη βέλτιστη αρχιτεκτονική CNN η απόδοση που πετυχαίνουμε είναι πολύ μικρή. Αυτό είναι λογικό, καθώς το δίκτυο είναι προσαρμοσμένο στις ανάγκες ενός τελείως διαφορετικού dataset.





Βλέποντας ενδεικτικά μία από τις εικόνες που έχουμε να ταξινομήσουμε βλέπουμε ότι δεν έχει καθόλου ίδια χαρακτηριστικά με το αρχικό dataset όπου οι εικόνες απεικόνιζαν χέρια. Εκεί μας ενδιέφερε κυρίως το περίγραμμα του χεριού (πολύ πιο «χοντροκομένα» πράγματα απ’ ότι εδώ, όπου η διαφορά των εικόνων φαίνεται με το ζόρι να διακρίνεται). Επιπλέον, τώρα μας ενδιαφέρει και το χρώμα άρα το grayscale δεν είναι ιδανικό, θα σύμφερε επίσης να κάνουμε horizontal και vertical flip τις εικόνες και πιθανότατα θα χρειαζόταν περισσότερα layers για την ανίχνευση τόσο δύσκολων χαρακτηριστικών.



Class 0

Class 1

Με την υλοποίηση του ResNet50 παρατηρούμε ότι τα αποτελέσματα είναι μακράν καλύτερα:

```
Epoch 1/500
164/164 [==============================] - 630s 4s/step - loss: 1.3036 - accuracy: 0.6833 - val_loss: 0.8951 - val_accuracy: 0.7395
Epoch 2/500
164/164 [==============================] - 375s 2s/step - loss: 0.7083 - accuracy: 0.7584 - val_loss: 0.8152 - val_accuracy: 0.7449
Epoch 3/500
164/164 [==============================] - 379s 2s/step - loss: 0.6664 - accuracy: 0.7731 - val_loss: 0.8636 - val_accuracy: 0.7428
Epoch 4/500
164/164 [==============================] - 377s 2s/step - loss: 0.6230 - accuracy: 0.7876 - val_loss: 0.8338 - val_accuracy: 0.7396
Epoch 5/500
164/164 [==============================] - 377s 2s/step - loss: 0.5734 - accuracy: 0.8019 - val_loss: 0.7707 - val_accuracy: 0.7581
Epoch 6/500
164/164 [==============================] - 379s 2s/step - loss: 0.5246 - accuracy: 0.8166 - val_loss: 0.8868 - val_accuracy: 0.6961
Epoch 7/500
164/164 [==============================] - 379s 2s/step - loss: 0.4562 - accuracy: 0.8389 - val_loss: 1.0363 - val_accuracy: 0.7483
Epoch 8/500
164/164 [==============================] - 377s 2s/step - loss: 0.3627 - accuracy: 0.8658 - val_loss: 1.2834 - val_accuracy: 0.7550
Epoch 9/500
164/164 [==============================] - 377s 2s/step - loss: 0.2851 - accuracy: 0.8935 - val_loss: 1.0798 - val_accuracy: 0.7451
Epoch 10/500
164/164 [==============================] - 377s 2s/step - loss: 0.2160 - accuracy: 0.9218 - val_loss: 1.1531 - val_accuracy: 0.7490
Epoch 11/500
164/164 [==============================] - 378s 2s/step - loss: 0.1716 - accuracy: 0.9411 - val_loss: 1.1389 - val_accuracy: 0.7248
Epoch 12/500
 13/164 [=>............................] - ETA: 5:01 - loss: 0.1209 - accuracy: 0.9574
```

Δυστυχώς δεν πρόλαβε να ολοκληρωθεί η υλοποίηση μέχρι την ώρα παράδοσης της εργασίας, φαίνεται όμως από το val_accuracy ότι τα αποτελέσματα είναι πάρα πολύ ενθαρυντικά.

Το ResNet-50 είναι ένα convolutional neural network που αποτελείται από 50 layers.

```
model = tf.keras.applications.ResNet50(
    include_top=False,
    input_tensor=None,
    input_shape=((100, 100, 3)),
    pooling=None,
    classes=num_classes
)
```

b. Ποιες είναι οι διαφορές ενός CNN με το ResNet;

Το ResNet περιέχει skip connections και πολύ batch normalization. Τα skip connections είναι επίσης γνωστά ως gated units και επιτρέπουν στα gradients να κυλάν ομαλά από layer σε layer. Αυτό που κάνουν είναι να παραλείπουν κάποιο layer του νευρωνικού δικτύου και να δώσουν τις εξόδους του προηγούμενού του layer σε επόμενά του layers. Με τον τρόπο αυτό από την πολύ αρχή ακόμη και το τελευταίο layer λαμβάνει βάρη από το top layer.
Τα skip connections είναι ουσιαστικά η κύρια διαφορά των 2 ειδών δικτύων.

c. Ποιο ήταν το κύριο πρόβλημα που αποφεύχθηκε κατά την εκπαίδευση βαθέων νευρωνικών δικτύων με το ResNet;

Χρησιμοποιώντας τον κανόνα της αλυσίδας, πρέπει να συνεχίσουμε να πολλαπλασιάζουμε τους όρους με την κλίση σφάλματος καθώς πηγαίνουμε προς τα πίσω (στη διαδικασία back propagation, η οποία είναι απαραίτητη για την εκπαίδευση του δικτύου).
Ωστόσο, στη μακρά αλυσίδα πολλαπλασιασμού, εάν πολλαπλασιάσουμε πολλούς αριθμούς μαζί που είναι μικρότεροι από ένα, τότε η προκύπτουσα κλίση θα είναι πολύ μικρή. Έτσι, η κλίση γίνεται πολύ μικρή καθώς πλησιάζουμε τα πιο αρχικά στρώματα σε μια βαθιά αρχιτεκτονική. Σε ορισμένες περιπτώσεις, η κλίση γίνεται μηδέν, πράγμα που σημαίνει ότι δεν ενημερώνουμε καθόλου τα πρώτα επίπεδα. Αυτό ακριβώς το πρόβλημα λύνει το ResNet με την προσθήκη των skip connections που αναφέρθηκαν προηγουμένως.
Τώρα η σχέση ενός νευρώνα του τελευταίου επιπέδου με έναν που βρισκόταν στην πολύ αρχή δεν συμπεριλαμβάνει απαραίτητα τις σχέσεις των νευρώνων που υπάρχουν στα ενδιάμεσα επίπεδα.