Βαθιά Μάθηση και Ανάλυση Πολυμεσικών Δεδομένων

Ευαγγελία Κυριακοπούλου

ΑΕΜ : 110

Contents:

1. Introduction

Για την εργασία επιλέχθηκε ένα dataset για Facial Emotion Recognition μέσω του Kaggle. Το dataset αυτό είναι μια συλλογή 35887 εικόνων, 48x48 pixel σε gray scale . Κάθε εικόνα απεικονίζει και ένα διαφορετικό συναίσθημα. Τα συναισθήματα που αποτυπώνονται στις εικόνες είναι: angry, disgust, fear, happy, neutral, sad και surprise. Συνεπώς, έχουμε να κάνουμε με ένα Multi – Class Image Classification πρόβλημα, 7 κλάσεων, το οποίο και θα προσεγγίσουμε με χρήση Βαθιών Συνελικτικών Νευρωνικών Δικτύων (Deep Convolutional Neural Networks). Για την συγγραφή και την εκτέλεση του κώδικα έγινε χρήση του περιβάλλοντος Jupyter Notebook.

2. Importing necessary libraries

Το πρόγραμμα υλοποιήθηκε στη γλώσσα προγραμματισμού Python με χρήση των παρακάτω βιβλιοθηκών :

2. Importing necessary libraries

+ Code + Markdown

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import random
from tqdm.notebook import tqdm
warnings.filterwarnings('ignore')
%matplotlib inline

from PIL import Image
from sklearn.preprocessing import LabelEncoder
import torch.nn as nn
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.utils import load_img,plot_model
from keras.layers import Layer
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
```

Εικόνα 1 Φόρτωση Βιβλιοθηκών

3. Loading the data & 4. Data pre-processing

Μετά την εισαγωγή των βιβλιοθηκών ακολούθησε η φόρτωση των δεδομένων μέσω των paths για τα δεδομένα ελέγχου και εκπαίδευσης, όπως και η μετατροπή αυτών σε data frames. Το data set ήταν ήδη χωρισμένο σε δεδομένα ελέγχου και εκπαίδευσης οπότε δε χρειάστηκε να προβούμε σε split-άρισμα. Με την βοήθεια συνάρτησης για κάθε μία κλάση που ολοκληρωνόταν η μετατροπή, τυπωνόταν στην οθόνη το μήνυμα “label” Completed και τέλος ακολούθησε το πλοτάρισμα των δεδομένων εκπαίδευσης και ελέγχου όπως φαίνεται παρακάτω :

3.Loading the Data

4.Data pre-processing

```
train_dir = '../input/facial-expression-dataset/train/train/'
test_dir = '../input/facial-expression-dataset/test/test/'
```

+ Code + Markdown

```
def load_dataset(directory):
    image_paths = []
    labels = []

    for label in os.listdir(directory):
        for filename in os.listdir(directory+label):
            image_path = os.path.join(directory, label, filename)
            image_paths.append(image_path)
            labels.append(label)

        print(label, "Completed")

    return image_paths, labels
```

Εικόνα 2Φόρτωση Δεδομένων και βοηθητική συνάρτηση

```
## convert into dataframe
train = pd.DataFrame()
train['image'], train['label'] = load_dataset(train_dir)
# shuffle the train dataset
train = train.sample(frac=1).reset_index(drop=True)
train.head()
```

```
surprise Completed
fear Completed
angry Completed
neutral Completed
sad Completed
disgust Completed
happy Completed
```

	image	label
0	../input/facial-expression-dataset/train/train...	neutral
1	../input/facial-expression-dataset/train/train...	sad
2	../input/facial-expression-dataset/train/train...	happy
3	../input/facial-expression-dataset/train/train...	angry
4	../input/facial-expression-dataset/train/train...	surprise

Εικόνα 3 Μετατροπή train set σε data frame & shuffle

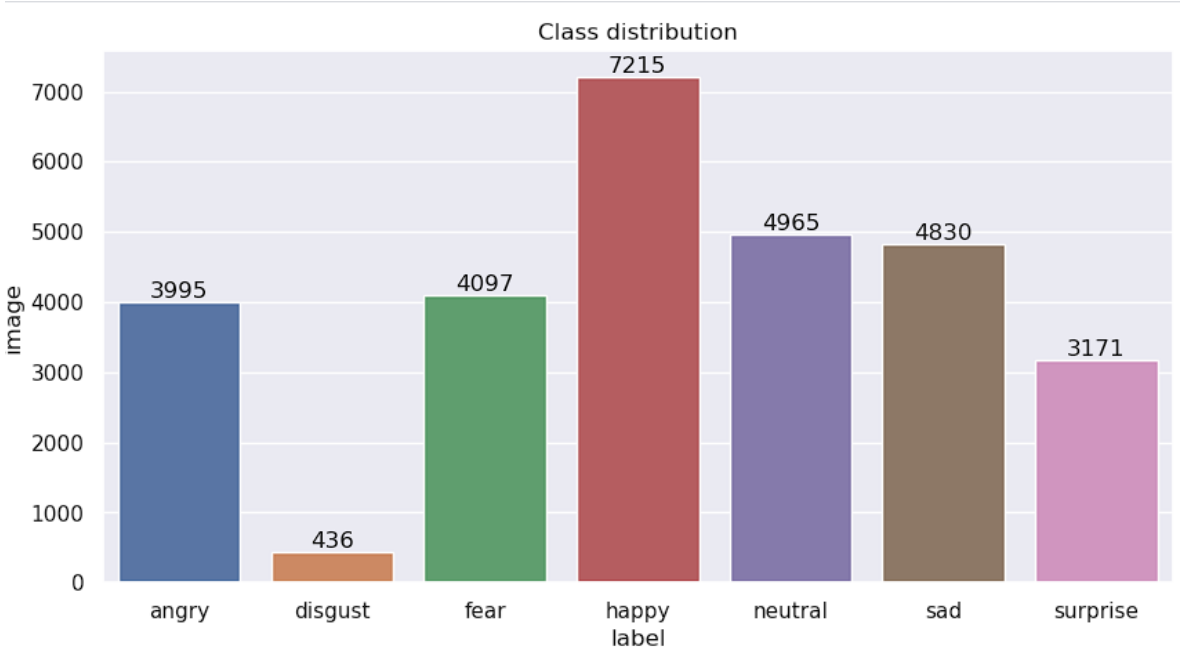
```
5]:
test = pd.DataFrame()
test['image'], test['label'] = load_dataset(test_dir)
test.head()
```

```
surprise Completed
fear Completed
angry Completed
neutral Completed
sad Completed
disgust Completed
happy Completed
```

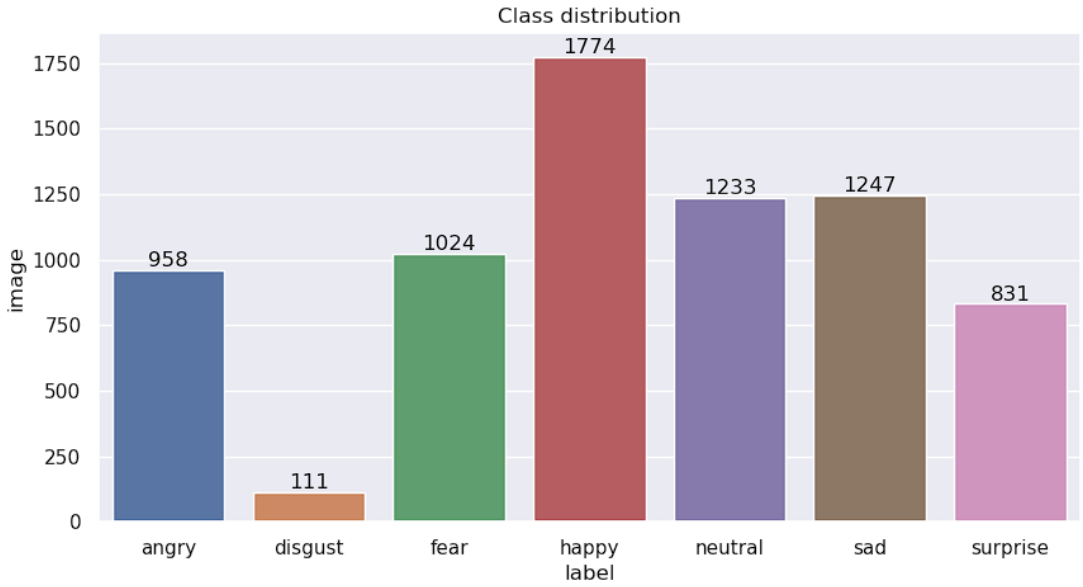
6]:

	image	label
0	../input/facial-expression-dataset/test/test/s...	surprise
1	../input/facial-expression-dataset/test/test/s...	surprise
2	../input/facial-expression-dataset/test/test/s...	surprise
3	../input/facial-expression-dataset/test/test/s...	surprise
4	../input/facial-expression-dataset/test/test/s...	surprise

Εικόνα 4 Μετατροπή test set σε data frame



Εικόνα 5 Train set



Εικόνα 6 Test set

Στο train set έχουμε :

Label	Angry	Disgust	Fear	Happy	Neutral	Sad	Surprise
Number of Images	3995	436	4097	7215	4965	4830	3171

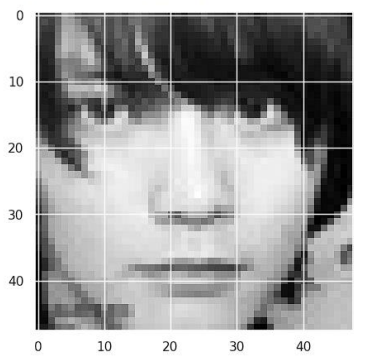
Ενώ στο test set έχουμε :

Label	Angry	Disgust	Fear	Happy	Neutral	Sad	Surprise
Number of Images	958	111	1024	1774	1233	1247	831

Παρακάτω βλέπουμε την 3η εικόνα από το train και την 300η εικόνα από το test set αντίστοιχα

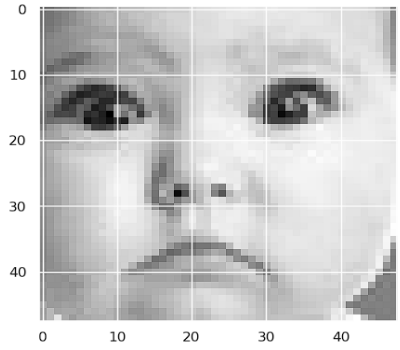
```
img = Image.open(train['image'][3])
plt.imshow(img, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x74961e1536d0>
```



```
img = Image.open(test['image'][300])
plt.imshow(img, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x74961e19a910>
```



Για να πάρουμε μια ιδέα πώς είναι οι εικόνες, θα πάρουμε τις πρώτες 25 από αυτές από το train set κάνοντας χρήση του κώδικα όπως φαίνεται παρακάτω :

```
# to display grid of images
plt.figure(figsize=(8,8))
files = train.iloc[0:25]

for index, file, label in files.itertuples():
    plt.subplot(5, 5, index +1)
    img = load_img(file)
    img = np.array(img)
    plt.imshow(img)
    plt.title(label)
    plt.axis('off')
```

Εικόνα 7 25 πρώτες εικόνες από train set

Επιπρόσθετα, ακολούθησε η μετατροπή των εικόνων σε numpy arrays και ο μετασχηματισμός αυτών έχοντας τώρα νέες διαστάσεις 48x48x1, αντί 48x48. Στα νευρωνικά δίκτυα επίσης δεν αρέσουν τα

δεδομένα με μεγάλες διακυμάνσεις, οπότε τα μετατρέψαμε ώστε να έχουν τιμές μεταξύ 0 και 1. Παρακάτω φαίνεται ο κώδικας για όσο περιεγράφηκαν παραπάνω :

```
def extract_features(images):
    features = []
    for image in tqdm(images):
        img = load_img(image, grayscale=True)
        img = np.array(img)
        features.append(img)
    features = np.array(features)
    features = features.reshape(len(features), 48, 48, 1)
    return features
```

```
train_features = extract_features(train['image'])
```

```
100% 28709/28709 [04:07<00:00, 113.30it/s]
```

```
test_features = extract_features(test['image'])
```

```
100% 7178/7178 [00:53<00:00, 119.96it/s]
```

+ Code + Markdown

```
## normalize the image
x_train = train_features/255.0
x_test = test_features/255.0
```

Η τελευταία επεξεργασία που εφαρμόστηκε στα δεδομένα πριν την δημιουργία του συνελικτικού νευρωνικού δικτύου ήταν η μετατροπή των labels σε integers εφαρμόζοντας One - Hot - Encoding. Ο στόχος μας είναι να φτιάξουμε διανύσματα με 7 αριθμούς, π.χ. αν η εικόνα αντιστοιχεί σε συναίσθημα happy: "[0, 0, 0, 0, 0, 0, 1]". Κάθε ένα από αυτά τα διανύσματα έχει μία τιμή ίση με 1 (η σωστή κλάση), ενώ οι άλλες τιμές είναι ίσες με 0. Αυτό σημαίνει ότι κάθε στόχος είναι στην πραγματικότητα μια κατανομή πιθανοτήτων, αφού είναι ένα σύνολο τιμών από το μηδέν έως το ένα που αθροίζουν στο ένα.

```
## convert label to integer
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(train['label'])
y_train = le.transform(train['label'])
y_test = le.transform(test['label'])
```

```
y_train = to_categorical(y_train, num_classes=7)
y_test = to_categorical(y_test, num_classes=7)
```

```
y_train[0]
```

```
array([0., 0., 0., 0., 0., 0., 1.], dtype=float32)
```

5. Design a model

Πριν την δημιουργία του πρώτου CNN είναι σημαντικό να αναφερθεί ότι η συνάρτηση ενεργοποίησης Softmax που χρησιμοποιήθηκε κατά την εκπαίδευση του πρώτου, αλλά και στα επόμενα μοντέλα, γράφτηκε from scratch. Μαθηματικά η συνάρτηση ορίζεται ως εξής : αν έχουμε K κλάσεις, και ο k νευρώνας έχει έξοδο z_k, η συνάρτηση Softmax ορίζεται ως : $\sigma(z_k) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$, j=1,2,…,k . Στην ουσία, η Softmax δίνει τις εισόδους στην εκθετική συνάρτηση και κανονικοποιεί. Η εκθετική συνάρτηση σημαίνει ότι μια μονάδα αύξηση στην είσοδο έχει πολλαπλασιαστικά αποτελέσματα στην έξοδο. Αντιστρόφως, αν μειώνεται η είσοδος, στην έξοδο παίρνουμε κλάσμα της εισόδου. Η Softmax διαιρεί με το άθροισμα της εκθετικής συνάρτησης στις εισόδους, ώστε το αποτέλεσμα να είναι μεταξύ μηδέν και ένα.

Αρχικά ορίζεται μια κλάση επιπέδου Softmax που κληρονομείται-καλείται από την κλάση Keras Layer.

Η μέθοδος __init__ αρχικοποιεί το επίπεδο Softmax λαμβάνοντας τον αριθμό των κλάσεων ως είσοδο και αποθηκεύοντάς τον ως μεταβλητή εμφάνισης self.num_classes. Η συνάρτηση super καλείται για να αρχικοποιήσει το επίπεδο γονικής κλάσης. Η μέθοδος build σε αυτήν την κλάση επιπέδων Softmax είναι υπεύθυνη για τη δημιουργία των εκπαιδεύσιμων βαρών του επιπέδου.

Η μέθοδος build καλείται μία φορά κατά την προετοιμασία του στρώματος και μεταβιβάζεται το σχήμα του τανυστή εισόδου που θα περάσει μέσα από το επίπεδο. Η μέθοδος self.add_weight () καλείται δύο φορές για να δημιουργήσει δύο εκπαιδεύσιμες μεταβλητές για το επίπεδο: έναν πίνακα βάρους Weights και ένα διάνυσμα πόλωσης biases. Ο πίνακας βάρους Weights έχει σχήμα (input_shape[-1], self.num_classes), όπου input_shape[-1] είναι το μέγεθος της τελευταίας διάστασης του τανυστή εισόδου και self.num_classes είναι ο αριθμός των κλάσεων εξόδου. Ο πίνακας βάρους αρχικοποιείται χρησιμοποιώντας τον αρχικοποιητή «random_normal», ο οποίος δημιουργεί τυχαίους αριθμούς από μια

κανονική κατανομή με μέσο όρο 0 και τυπική απόκλιση 1. Το διάνυσμα πόλωσης biases έχει σχήμα (self.num_classes,), το οποίο αντιστοιχεί στον αριθμό των κλάσεων εξόδου. Το διάνυσμα πόλωσης αρχικοποιείται χρησιμοποιώντας τον αρχικοποιητή «μηδενικά». Τόσο το Weights όσο και το biases δημιουργούνται ως μεταβλητές στιγμιότυπου του επιπέδου Softmax, γεγονός που τις καθιστά εκπαιδεύσιμες από τον βελτιστοποιητή κατά τη διάρκεια της εκπαιδευτικής διαδικασίας.

Στη συνέχεια ακολουθεί η προς τα εμπρός μέθοδος (forward pass) . Η forward μέθοδος σε αυτήν την κλάση επιπέδου Softmax είναι υπεύθυνη για τον καθορισμό του μπροστινού περάσματος του επιπέδου. Η μέθοδος forward παίρνει έναν τανυστή εισόδου και εφαρμόζει έναν γραμμικό μετασχηματισμό που ακολουθείται από τη συνάρτηση Softmax για τον υπολογισμό του τανυστή εξόδου του στρώματος. Η μέθοδος αυτή λαμβάνει εισόδους τανυστή εισόδου και εκτελεί έναν γραμμικό μετασχηματισμό πολλαπλασιάζοντάς τον με τον πίνακα βάρους Weights και προσθέτοντας τις πολώσεις διανυσμάτων πόλωσης. Το αποτέλεσμα αυτού του υπολογισμού είναι ένα διάνυσμα logits, τα οποία είναι βαθμολογίες πραγματικών τιμών που υποδεικνύουν την πιθανότητα η είσοδος να ανήκει σε κάθε κλάση. Τα logit στη συνέχεια περνούν μέσω της συνάρτησης tf.nn.softmax, η οποία εφαρμόζει τη συνάρτηση ενεργοποίησης Softmax στα logit για να τα μετατρέψει σε κατανομή πιθανότητας στις κλάσεις εξόδου. Η συνάρτηση Softmax διασφαλίζει ότι οι έξοδοι του επιπέδου κανονικοποιούνται και αθροίζονται στη μονάδα, καθιστώντας τις ερμηνεύσιμες ως πιθανότητες κλάσης. Η έξοδος της forward είναι οι πιθανότητες Softmax στις κλάσεις εξόδου, οι οποίες επιστρέφονται ως τανυστής εξόδου του στρώματος.

Η compute_output_shape σε αυτήν την κλάση επιπέδου Softmax είναι υπεύθυνη για τον υπολογισμό του σχήματος εξόδου του στρώματος με βάση το σχήμα εισόδου. Η compute_output_shape απλώς επιστρέφει το ίδιο σχήμα εισόδου που λαμβάνει ως όρισμα. Αυτό συμβαίνει επειδή το επίπεδο Softmax δεν αλλάζει το σχήμα του τανυστή εισόδου, αλλά εφαρμόζει μόνο έναν γραμμικό μετασχηματισμό και μια συνάρτηση ενεργοποίησης Softmax σε αυτόν. Επιστρέφοντας το ίδιο σχήμα εισόδου, αυτή η μέθοδος διασφαλίζει ότι το στρώμα μπορεί να χρησιμοποιηθεί ως δομικό στοιχείο σε ένα μοντέλο Keras και ότι τα επόμενα στρώματα μπορούν να υπολογίσουν σωστά τα σχήματα εισόδου τους με βάση τα σχήματα εξόδου των προηγούμενων στρωμάτων.

Τέλος, έχουμε την get_config στην τελευταία κλάση επιπέδου Softmax, η οποία είναι υπεύθυνη για την επιστροφή ενός λεξικού παραμέτρων διαμόρφωσης που μπορεί να χρησιμοποιηθεί για την αναδημιουργία της παρουσίας του επιπέδου σε μεταγενέστερο χρόνο. Η μέθοδος get_config καλεί πρώτα τη μέθοδο get_config της γονικής κλάσης Layer χρησιμοποιώντας τη συνάρτηση super και αποθηκεύει το λεξικό διαμόρφωσης που προκύπτει στη μεταβλητή διαμόρφωσης. Στη συνέχεια, η μέθοδος προσθέτει το χαρακτηριστικό num_classes του επιπέδου στο λεξικό διαμόρφωσης, ώστε να μπορεί να συμπεριληφθεί στην αποθηκευμένη διαμόρφωση. Τέλος, η μέθοδος επιστρέφει το ενημερωμένο λεξικό διαμόρφωσης, το οποίο μπορεί να χρησιμοποιηθεί για την αναδημιουργία της παρουσίας του επιπέδου καλώντας τη μέθοδο κλάσης from_config του επιπέδου, περνώντας το λεξικό διαμόρφωσης ως όρισμα. Παρακάτω βλέπουμε τον σχετικό κώδικα.

```
num_classes=7
from keras.layers import Layer
import torch.nn as nn

class Softmax(Layer):

    def __init__(self, num_classes):
        super(Softmax, self).__init__()
        self.num_classes = num_classes

    def build(self, input_shape):
        self.Weights = self.add_weight(name='Weights',
                                       shape=(input_shape[-1], self.num_classes),
                                       initializer='random_normal',
                                       trainable=True)
        self.biases = self.add_weight(name='biases',
                                      shape=(self.num_classes,),
                                      initializer='zeros',
                                      trainable=True)

    def call(self, inputs):
        logits = tf.matmul(inputs, self.Weights) + self.biases
        return tf.nn.softmax(logits)

    def compute_output_shape(self, input_shape):
        return input_shape

    def get_config(self):
        config = super(Softmax, self).get_config()
        config['num_classes'] = self.num_classes
        return config
```

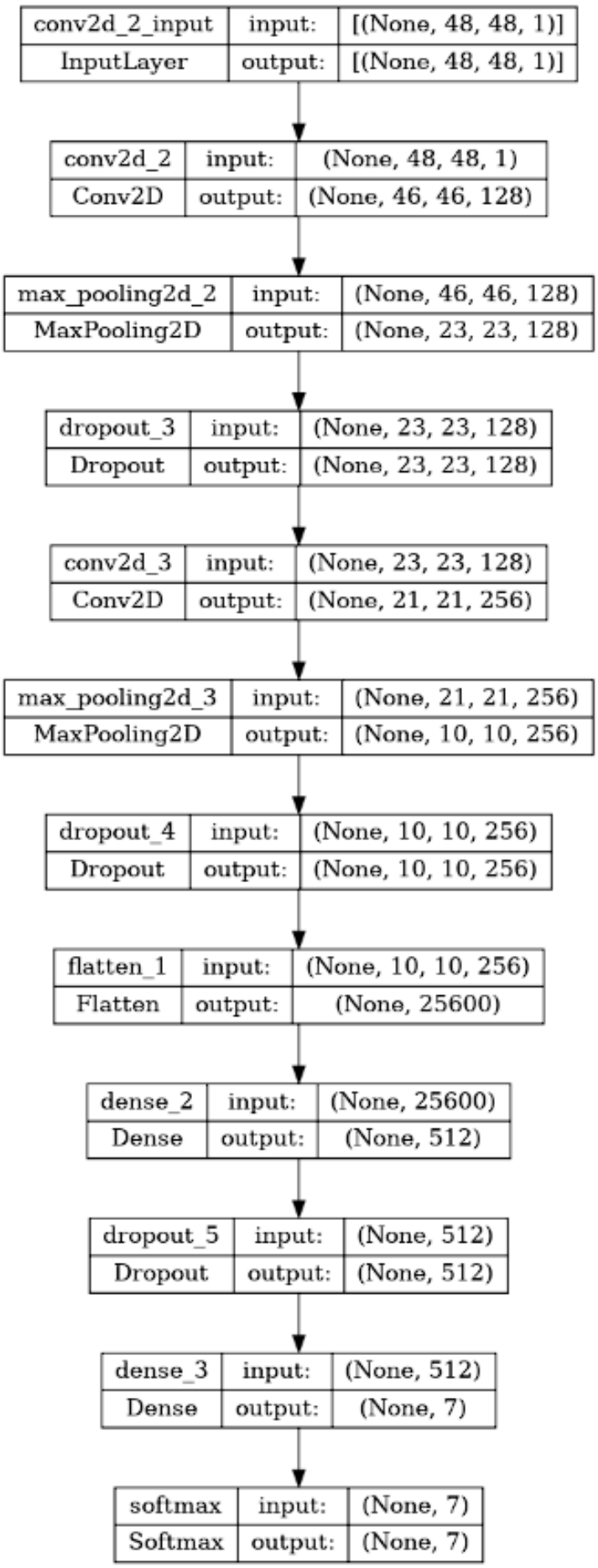
Εικόνα 8 Softmax code

Αφού αρχικοποιήθηκαν κάποιες παράμετροί, δημιουργήθηκε το πρώτο συνελικτικό μοντέλο και χρησιμοποιήθηκε για την εκπαίδευση. Το μοντέλο φαίνεται παρακάτω :

```
#initialize parameters
width, height = (48, 48)
output_class = 7
```

```
model = Sequential()

# convolutional layers

#model 1
model.add(Conv2D(128, kernel_size=(3,3), activation='relu', input_shape=(width, height, 1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.1))

#model 2
model.add(Conv2D(256, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))


#flatten
model.add(Flatten())

# fully connected layers
#dense 1
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))

# output layer
model.add(Dense(7))
model.add(Softmax(num_classes=7))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics='accuracy')
```

Εικόνα 9 Αρχιτεκτονική Πρώτου Μοντέλου


conv2d_2_input | input: | [(None, 48, 48, 1)]
InputLayer | output: | [(None, 48, 48, 1)]
conv2d_2 | input: | (None, 48, 48, 1)
Conv2D | output: | (None, 46, 46, 128)
max_pooling2d_2 | input: | (None, 46, 46, 128)
MaxPooling2D | output: | (None, 23, 23, 128)
dropout_3 | input: | (None, 23, 23, 128)
Dropout | output: | (None, 23, 23, 128)
conv2d_3 | input: | (None, 23, 23, 128)
Conv2D | output: | (None, 21, 21, 256)
max_pooling2d_3 | input: | (None, 21, 21, 256)
MaxPooling2D | output: | (None, 10, 10, 256)
dropout_4 | input: | (None, 10, 10, 256)
Dropout | output: | (None, 10, 10, 256)
flatten_1 | input: | (None, 10, 10, 256)
Flatten | output: | (None, 25600)
dense_2 | input: | (None, 25600)
Dense | output: | (None, 512)
dropout_5 | input: | (None, 512)
Dropout | output: | (None, 512)
dense_3 | input: | (None, 512)
Dense | output: | (None, 7)
softmax | input: | (None, 7)
Softmax | output: | (None, 7)

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 46, 46, 128)       1280

 max_pooling2d_2 (MaxPooling  (None, 23, 23, 128)      0
 2D)

 dropout_3 (Dropout)         (None, 23, 23, 128)       0

 conv2d_3 (Conv2D)           (None, 21, 21, 256)       295168

 max_pooling2d_3 (MaxPooling  (None, 10, 10, 256)      0
 2D)

 dropout_4 (Dropout)         (None, 10, 10, 256)       0

 flatten_1 (Flatten)         (None, 25600)             0

 dense_2 (Dense)             (None, 512)               13107712

 dropout_5 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 7)                 3591

 softmax (Softmax)           (None, 7)                 56

=================================================================
Total params: 13,407,807
Trainable params: 13,407,807
Non-trainable params: 0
_________________________________________________________________
```

Αναλυτικά η αρχιτεκτονική που ακολουθήθηκε ήταν η εξής : Φτιάξαμε μια συνελικτική βάση που αποτελείται από μία σειρά στρωμάτων `Conv2D` , `MaxPooling2D` και `Dropout`. Η είσοδος στο νευρωνικό δίκτυο θα είναι 48,48, 1, αφού έχουμε gray scale χρώματα.

Convolutional Layers

- Το πρώτο συνελικτικό στρώμα περιέχει 128 φίλτρα. Το κάθε ένα από τα οποία είναι διαστάσεων 3x3. Η έξοδος είναι (46,46,128), αφού αν το σκεφτούμε ένα φίλτρο 3x3 χωράει 46 φορές οριζόντια και κάθετα σε μία εικόνα 48x48. Το στρώμα MaxPooling2D θα μειώσει τις διαστάσεις στο μισό και έτσι τώρα θα έχουμε (23,23,128). Με την μέθοδο dropout θερίζουμε από το νευρωνικό δίκτυο τους κόμβους με πιθανότητα p=0.1. Επίσης η συνάρτηση ενεργοποίησης που καλείται στο επίπεδο είναι η relu.

- Το δεύτερο συνελικτικό στρώμα περιέχει 256 φίλτρα. Το κάθε ένα από τα οποία είναι διαστάσεων 3x3. Η έξοδος είναι (21,21,256). Το στρώμα MaxPooling2D θα μειώσει τις διαστάσεις στο μισό και έτσι τώρα θα έχουμε (10,10,256). Εδώ αντίστοιχα θερίζουμε από το

νευρωνικό δίκτυο τους κόμβους με πιθανότητα p=0.4. Επίσης η συνάρτηση ενεργοποίησης που καλείται στο επίπεδο είναι η relu.

Flatten Layer

- Στο Flatten στρώμα θα μετατρέψουμε την έξοδο (10,10,256) σ ’έναν μονοδιάστατο πίνακα διαστάσεων 10x10x256. Το αποτέλεσμα θα το περάσουμε από ένα πυκνό στρώμα με 25600 νευρώνες.

Dense Layer

- Στη συνέχεια έχουμε ένα κρυμμένο στρώμα εισόδου με 512 νευρώνες και ένα κρυμμένο στρώμα εξόδου με 7 νευρώνες. Στο πρώτο κρυμμένο στρώμα εισόδου η συνάρτηση ενεργοποίησης που καλείται είναι η relu, ενώ στο στρώμα εξόδου δεν καλείται κάποια συνάρτηση ενεργοποίησης. Τέλος προστίθεται ένα ακόμα στρώμα στο συνελικτικό δίκτυο για το κάλεσμα της συνάρτησης Softmax.

Μπορούμε επίσης να επιβεβαιώσουμε τον αριθμό των παραμέτρων : Για το πρώτο συνελικτικό στρώμα έχουμε ((σχήμα πλάτους φίλτρου*σχήμα φίλτρου ύψους*αριθμός φίλτρων στην προηγούμενη στρώση+1)*αριθμός φίλτρων) = (((3*3*128)+1)*128) = 1280.

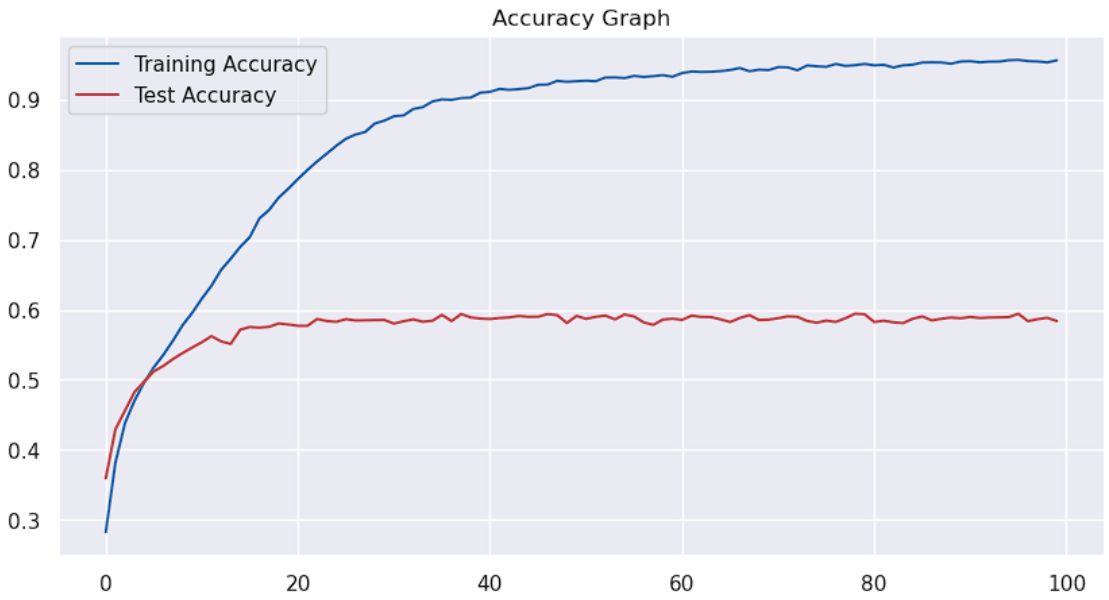
Για το δεύτερο συνελικτικό στρώμα έχουμε ((σχήμα πλάτους φίλτρου*σχήμα φίλτρου ύψους*αριθμός φίλτρων στην προηγούμενη στρώση+1)*αριθμός φίλτρων) = (((3*3*128)+1)*256) = 295168.

Για το πρώτο κρυμμένο στρώμα εισόδου έχουμε ((τρέχον στρώμα c *προηγούμενο στρώμα p)+1*c) = 512*25600+1*512= 13107712. Δηλαδή έχουμε 512 νευρώνες με 25601 βάρη και μια πόλωση, 512*25600+1*512= 13107712.

Στο κρυμμένο στρώμα εξόδου ο αριθμός των παραμέτρων ορίζεται πάλι ως εξής ((τρέχον στρώμα c *προηγούμενο στρώμα p)+1*c) = 7*512+1*7= 3591. Δηλαδή έχουμε 7 νευρώνες με 512 βάρη και μια πόλωση, 7*(512+1)=3591.

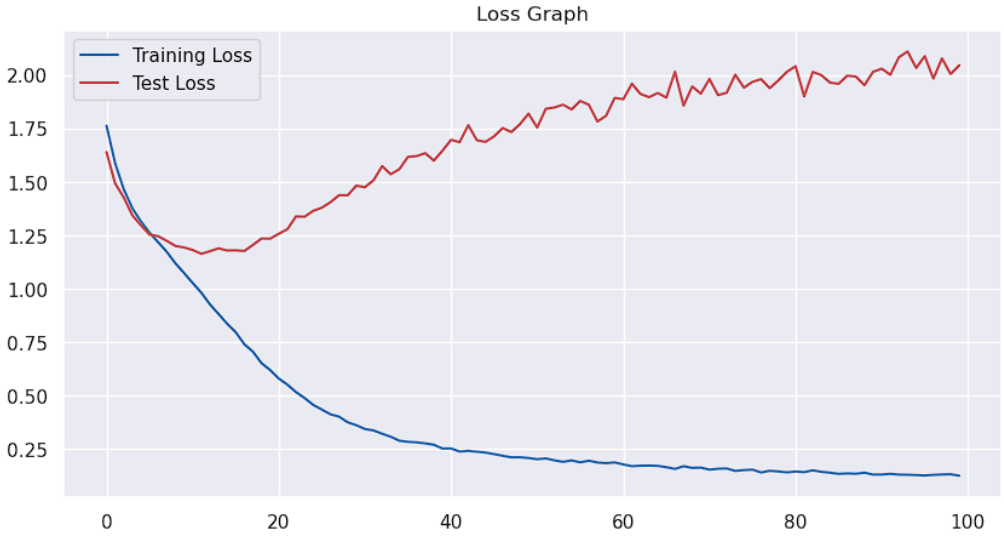
Τέλος για το στρώμα που καλείται η Softmax ο υπολογισμός γίνεται με τον ίδιο τρόπο όπως και στα κρυμμένα στρώματα, άρα έχουμε 7 νευρώνες με 7 βάρη και μια πόλωση 7*(7+1)=56 παραμέτρους.

Το μοντέλο εκπαιδεύτηκε με 256 batches σε 100 εποχές. Έπειτα έγινε προβολή μέσω διαγραμμάτων του accuracy και του loss του train και test set αντίστοιχα. Τα αποτελέσματα φαίνονται παρακάτω :



Εικόνα 10 Train & Test accuracy

Από την παραπάνω εικόνα μπορούμε να συμπεράνουμε το εξής: Τα δύο σετ δεδομένων φαίνεται μέχρι περίπου λίγο πριν την 10η εποχή να εκπαιδεύονται σωστά και το μοντέλο να μαθαίνει εξίσου καλά τα δεδομένα. Το υψηλότερο accuracy των δυο σετ μπορούμε να πούμε ότι αγγίζει το 50% . Στη συνέχεια, με το πέρας της 10ης εποχής είναι σαφής η ένδειξη υπερπροσαρμογής, καθώς το training accuracy αυξάνεται σταδιακά αγγίζοντας το 90%, ενώ το test accuracy αυξάνεται και παραμένει σταθερό χωρίς να ξεπερνά ποτέ το 60%. Όμοια συμπεράσματα παρατηρούνται και μέσω του διαγράμματος του loss, καθώς έχουμε σταδιακή πτώση του training loss και απότομη αύξηση του test loss.



Εικόνα 11 Train & Test Loss

6.Fine Tuning

Δοκιμάστηκαν διαφορετικές αρχιτεκτονικές δικτύων με περισσότερα convolution και dense layers. Επιπλέον δοκιμάστηκαν και άλλοι βελτιστοποιητές όπως ο Nadam, Ftrl (Follow The Regularized Leader) και ο SGD (Stochastic Gradient Descent). Αρχικά χρησιμοποιήθηκαν με default τιμές και μετά ακολούθησαν αυξομειώσεις στους ρυθμούς μάθησης. Ακόμη δημιουργήθηκε ένα learning rate schedule έτσι ώστε το δίκτυο αρχικά να μαθαίνει γρήγορα και στη συνέχεια όσο προχωρούσε η εκπαίδευση του να γίνεται πιο προσεκτικό. Χρησιμοποιώντας την κλάση `InverseTimeDecay` ορίσαμε ότι θα ξεκινήσουμε με έναν αρχικό ρυθμό συνέχειας και στις 10 εποχές αυτός θα μειώνεται ώστε στις 10 εποχές να γίνει 1/2 του αρχικού, στις 20 το 1/3 του αρχικού, κ.ο.κ. Επιπλέον δοκιμάστηκε και η συνάρτηση ενεργοποίησης ELU, με διάφορα παντρέματα με την RELU μεταξύ των layers. Σε όλες τις προσπάθειες τα αποτελέσματα ήταν είτε παρόμοια με το προηγούμενο μοντέλο είτε ακόμα χειρότερα με το μοντέλο να οδηγείται κατευθείαν σε υπερπροσαρμογή. Μετά από υπεράνθρωπες προσπάθειες το καλύτερο μοντέλο που καταφέραμε να φτιάξουμε παρουσιάζεται παρακάτω :

Πιο συγκεκριμένα προστέθηκαν άλλα δυο συνελικτικά επίπεδα και ένα ακόμα κρυφό επίπεδο. Έγινε αλλαγή της πιθανότητας σε όλα τα συνελικτικά και στα κρυφά επίπεδά και πλέον ισούται με p=0.4 , εκτός από το τελευταίο κρυφό στρώμα που ισούται πάλι με 0.3.

Convolutional Layers

- Το τρίτο συνελικτικό στρώμα περιέχει 512 φίλτρα. Το κάθε ένα από τα οποία είναι διαστάσεων 3x3. Η έξοδος είναι (8,8,512). Το στρώμα MaxPooling2D θα μειώσει τις διαστάσεις στο μισό και έτσι τώρα θα έχουμε (4,4,512). Με την μέθοδο dropout θερίζουμε από το νευρωνικό δίκτυο τους κόμβους με πιθανότητα p=0.4. Επίσης η συνάρτηση ενεργοποίησης που καλείται στο επίπεδο είναι η relu.

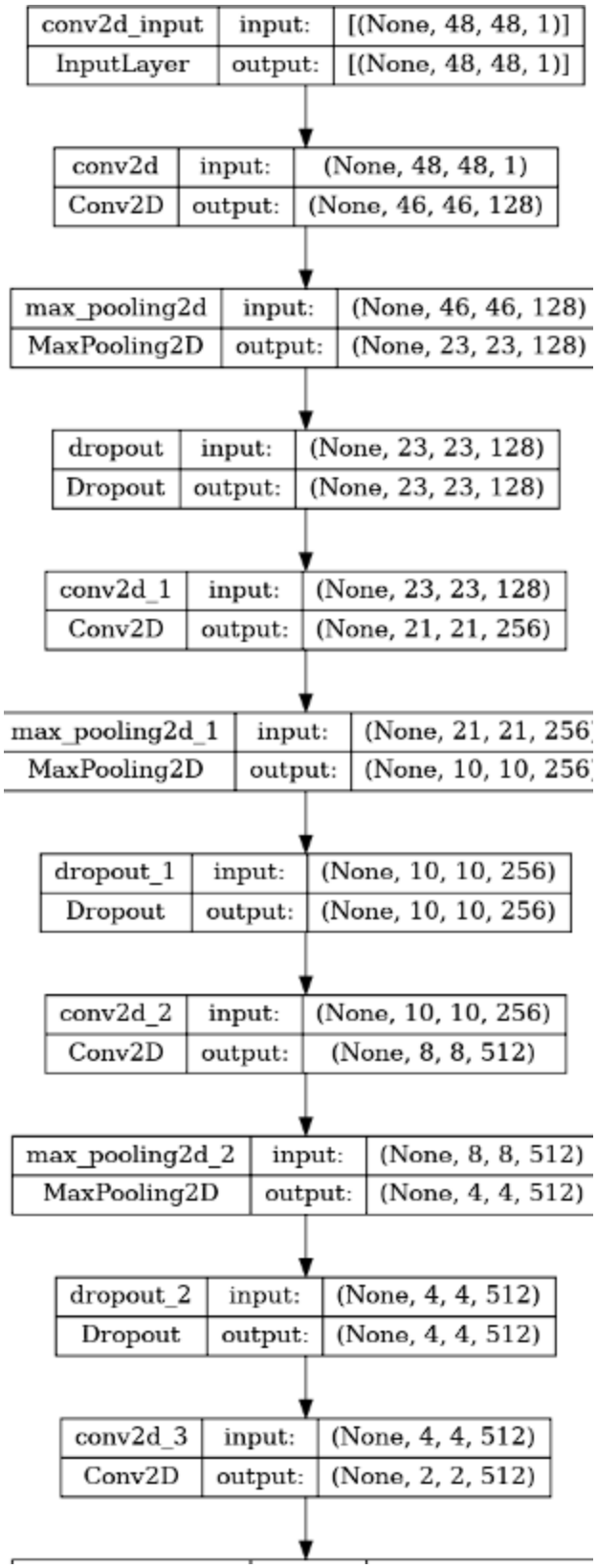
- Το τέταρτο συνελικτικό στρώμα περιέχει 512 φίλτρα. Το κάθε ένα από τα οποία είναι διαστάσεων 3x3. Η έξοδος είναι (2,2,512). Το στρώμα MaxPooling2D θα μειώσει τις διαστάσεις στο μισό και έτσι τώρα θα έχουμε (1,1,512). Εδώ αντίστοιχα θερίζουμε από το νευρωνικό δίκτυο τους κόμβους με πιθανότητα p=0.4. Επίσης η συνάρτηση ενεργοποίησης που καλείται στο επίπεδο είναι η relu.

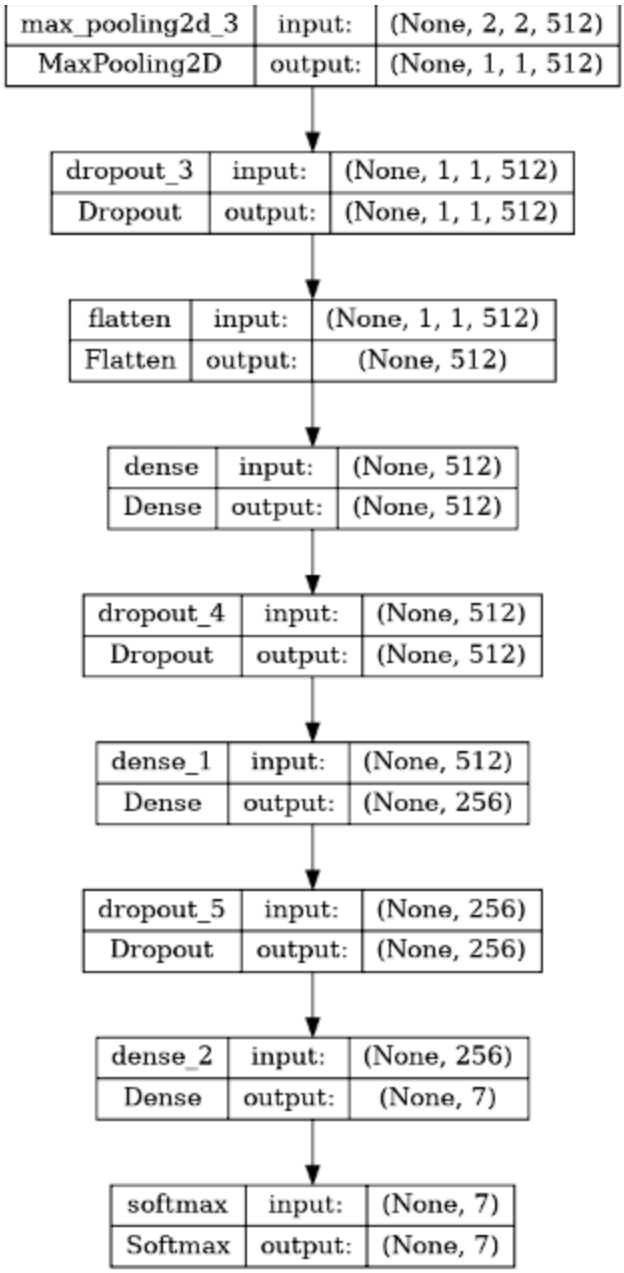
Dense Layer

- Στη συνέχεια για το δεύτερο κρυμμένο στρώμα έχουμε είσοδο 256 νευρώνων που συνδέονται με το πρώτο κρυμμένο στρώμα και η συνάρτηση ενεργοποίησης που καλείται είναι η relu και η πιθανότητα θερισμού στους κόμβους ισούται με p=0.3.

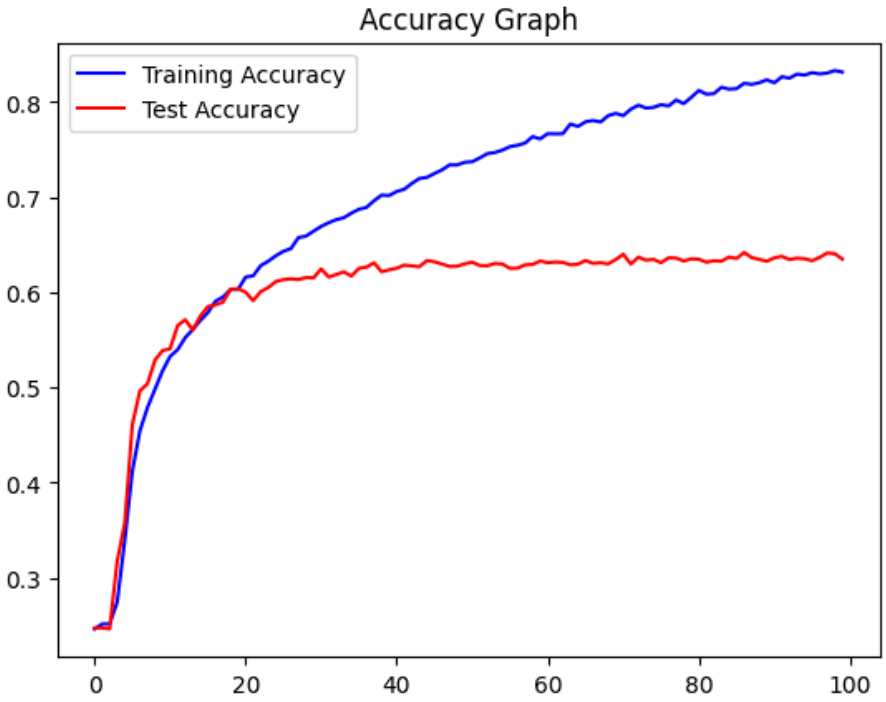
```
model = Sequential()

# convolutional layers

#model 1

model.add(Conv2D(128, kernel_size=(3,3), activation='relu', input_shape=(width, height, 1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

#model 2

model.add(Conv2D(256, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

#model 3

model.add(Conv2D(512, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

#model 4

model.add(Conv2D(512, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

#flatten

model.add(Flatten())

# fully connected layers

#dense 1

model.add(Dense(512, activation='relu'))
model.add(Dropout(0.4))


#dense 2
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.3))

# output layer

model.add(Dense(7))
model.add(Softmax(num_classes=7))
# model.add(Dense(output_class, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics='accuracy')
```

Εικόνα 12 Αρχιτεκτονική Δεύτερου Μοντέλου


conv2d_input | input: | [(None, 48, 48, 1)]
InputLayer | output: | [(None, 48, 48, 1)]
conv2d | input: | (None, 48, 48, 1)
Conv2D | output: | (None, 46, 46, 128)
max_pooling2d | input: | (None, 46, 46, 128)
MaxPooling2D | output: | (None, 23, 23, 128)
dropout | input: | (None, 23, 23, 128)
Dropout | output: | (None, 23, 23, 128)
conv2d_1 | input: | (None, 23, 23, 128)
Conv2D | output: | (None, 21, 21, 256)
max_pooling2d_1 | input: | (None, 21, 21, 256
MaxPooling2D | output: | (None, 10, 10, 256
dropout_1 | input: | (None, 10, 10, 256)
Dropout | output: | (None, 10, 10, 256)
conv2d_2 | input: | (None, 10, 10, 256)
Conv2D | output: | (None, 8, 8, 512)
max_pooling2d_2 | input: | (None, 8, 8, 512)
MaxPooling2D | output: | (None, 4, 4, 512)
dropout_2 | input: | (None, 4, 4, 512)
Dropout | output: | (None, 4, 4, 512)
conv2d_3 | input: | (None, 4, 4, 512)
Conv2D | output: | (None, 2, 2, 512)



Τα αποτελέσματα της εκπαίδευσης παρουσιάζονται στα παρακάτω δύο διαγράμματα



Εικόνα 13 Train & Test Loss accuracy

Εικόνα 14 Train & Test Loss

Από τις παραπάνω εικόνες μπορούμε να συμπεράνουμε το εξής: Τα δύο σετ δεδομένων φαίνεται μέχρι και την 20η εποχή να εκπαιδεύονται σωστά και το μοντέλο να μαθαίνει εξίσου καλά τα δεδομένα. Το

υψηλότερο accuracy των δυο σετ αγγίζει το 60% . 10% παραπάνω από ότι το προηγούμενο μοντέλο. Στη συνέχεια, με το πέρας της 23ης εποχής είναι σαφής η ένδειξη υπερπροσαρμογής, καθώς το training accuracy αυξάνεται σταδιακά αγγίζοντας το 83%, ενώ το test accuracy παραμένει σταθερό χωρίς να ξεπερνά ποτέ το 60%. Όμοια συμπεράσματα παρατηρούνται και μέσω του διαγράμματος του loss, καθώς έχουμε σταδιακή πτώση του training loss και απότομη αύξηση του test loss.

Ακολουθήθηκε έλεγχος από το test set για να δούμε και οπτικά το original output μιας εικόνας σε σχέση με το predicted label. Παρακάτω έχουμε μια πετυχημένη και μη πετυχημένη πρόβλεψη.