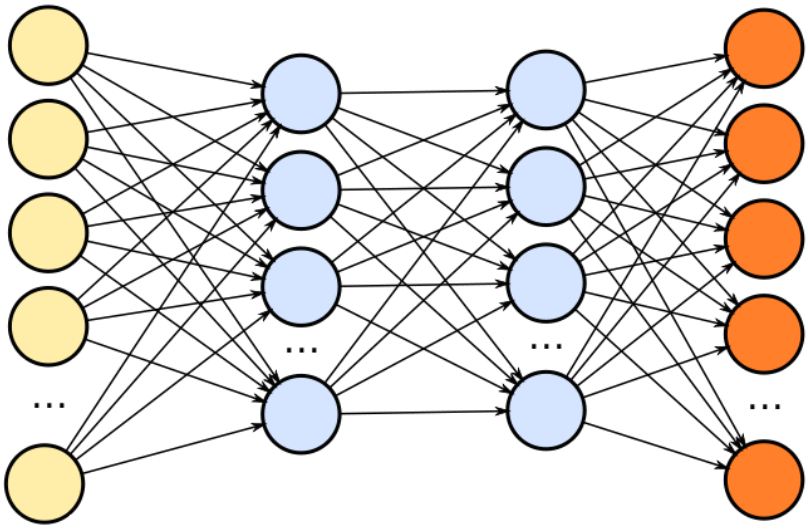
ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ | ΤΜΗΜΑ ΗΜ & ΜΥ

ΑΝΑΓΝΩΡΙΣΗ ΠΡΟΤΥΠΩΝ

Εξαμηνιαία Εργασία Εργαστηρίου
Καθηγητής: Μητιανούδης Νικόλαος
Υπεύθυνοι Εργαστηρίου: Γεώργιος-Αλέξης Ιωαννάκης
Βασιλάκης Ιωάννης



Μούσλεχ Στυλιανός Γεώργιος
ΑΜ: 57382
20/2/2021, Ξάνθη

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

Στην παρούσα εργασία θα ασχοληθούμε με την υλοποίηση διαφορετικών δομών νευρωνικών δικτύων για την ταξινόμηση εικόνων σε 2 συγκεκριμένα dataset.

Με βάση το επίθετό μου τα 2 dataset στα οποία θα δουλέψω είναι τα παρακάτω:

- Facial-Expression-Classification-Dataset(3Classes)

  Μια έκδοση του original FER(2013) Dataset με 3 κλάσεις και πρόσθετες εικόνες. Βρίσκεται στο παρακάτω link: (https://www.kaggle.com/nightfury007/fercustomdataset-3classes)

- Breast Histopathology Images

  Ένα dataset από εικόνες Ιστοπαθολογίας χωρισμένες ανά ασθενή και ανά θετικό και αρνητικό μέρος του ιστού (patch) σε Invasive Ductal Carcinoma. Βρίσκεται στο παρακάτω link: (https://www.kaggle.com/paultimothymooney/breast-histopathology-images)

Αρχικά γίνεται μια περιγραφή για την προετοιμασία του dataset (είναι σημαντικό να γίνει σωστά ώστε να έχουμε και ικανή επαναληψημότητα στα αποτελέσματα καθώς και σωστά αποτελέσματα)

Στη συνέχεια περιγράφονται οι υλοποιήσεις που ζητήθηκαν καθώς και οι απαντήσεις στις αντίστοιχες ερωτήσεις της εκφώνησης.

Τέλος γίνεται μια ανακεφαλαίωση των αποτελεσμάτων καθώς και μια κουβέντα για κάποιες δοκιμές που μου φαίνονται ενδιαφέρουσες αλλά δεν πρόλαβα να δοκιμάσω

Αναφορικά με τα αρχεία:

- dataPreProcessing_bonus.ipynb
  Το αρχείο της local προεπεξεργασίας του dataset για την δημιουργία train/test/val sets και αποθήκευσης numpy arrays των set αυτών
- ANN-Fully Connected FER(2013) Dataset.ipynb
  Η υλοποίηση των ερωτημάτων των fully connected δικτύων
- CNN-FER(2013) Dataset
  Η υλοποίηση των ερωτημάτων των CNN
- DataAugmentations_BEST_MODELipynb
  Η συνολικά καλύτερη υλοποίηση κάνοντας προ επεξεργασία των εικόνων (Histogram Equalization) και Data augmentations
- dataPreProcessing_bonus.ipynb
  Το αρχείο της local προεπεξεργασίας του bonus dataset για την δημιουργία train/test/val sets και αποθήκευσης numpy arrays των set αυτών
- Bonus.ipynb
  Η υλοποίηση των bonus ερωτημάτων
- Weights Folder
  Ένας φάκελος με τα βάρη από τα μοντέλα που γίναν. (τελικά λόγω χώρου δεν στάλθηκαν μπορώ να τα στείλω με mail αν χρειάζονται)

Περιγραφή Και Προετοιμασία Βασικού Dataset

Το αρχικό μας dataset περιέχει φωτογραφίες με ανθρώπινα πρόσωπα με σκοπό την ταξινόμηση των συναισθημάτων που εκφράζουν τα πρόσωπα αυτά.

Χωρίζεται σε 3 φακέλους , 1 για κάθε κλάση:

1. Disappointed (12734 εικόνες)
2. Interested (17210 εικόνες)
3. Neutral (13875 εικόνες)

Το dataset δεν είναι πολύ unbalanced ανά κλάση . (Η διαχείριση αυτής της διαφοροποίησης περιγράφεται παρακάτω.)

Το σύνολο των εικόνων είναι 43819 και υπάρχουν 2 ειδών εικόνων:

1. Grayscale εικόνες 48x48
2. RGB (έγχρωμες) εικόνες 64x64

(Ίσως από το αρχικό FER dataset και την προσθήκη εικόνων μετά στο Kaggle)

Με βάση τα παραπάνω, έπρεπε να γίνουν κάποιες «σχεδιαστικές» επιλογές για την διαχείριση του dataset με σκοπό να έχουμε σωστά αποτελέσματα και ένα "benchmark" dataset.

Οι προετοιμασία αυτή έγινε locally στο σύστημα μου.

Οι επιλογές είναι οι εξής:

Αποφάσισα να χωρίσω το dataset σε 3 φακέλους Training/Validation/Testing με τυχαίες φωτογραφίες από κάθε dataset κρατώντας φυσικά την αναλογία της κάθε κλάσης του dataset σε κάθε φάκελο που θα δημιουργηθεί .

Στη συνέχεια να φορτωθεί από τους φακέλους και να γίνουν όλες οι εικόνες ίδιου μεγέθους (κράτησα το 64x64) καθώς και να τις κάνω όλες grayscale.

Ο λόγος που προτίμησα να δουλέψω σε αυτό το dataset με grayscale εικόνες είναι διπλός:

1. Το χρώμα δεν αποτελεί σημαντική πληροφορία για το συναίσθημα που εκφράζει ένα πρόσωπο, μάλιστα μπορεί να είναι παραπλανητικό αν έχουμε συγκεκριμένα biases αν πχ κάποια χρώματα προσώπων είναι underrepresented σε συγκεκριμένες κλάσεις όταν το dataset δημιουργήθηκε.
2. Το dataset είναι μικτό . Αυτό σημαίνει ότι αν υπάρχουν ασσυμετρίες στην αναλογία έγχρωμης εικόνας με grayscale εικόνας ανά κλάση το μοντέλο μπορεί να «μάθει» αυτή τη πληροφορία και έτσι να θεωρείται παράμετρος τα χρώματα της εικόνας για τη κλάση . Αυτό το χαρακτηριστικό όμως δεν αντιπροσωπεύει την πραγματικότητα οπότε δεν είναι επιθυμητό

Τέλος αποθηκεύω σε numpy arrays το dataset και τα labels με σκοπό την εύκολη διαχείριση του και φόρτωση του στο google drive καθώς το training θα γίνεται μέσω του Google Collaboratory

Όλη η διαδικασία που περιγράφω εδώ υπάρχει στο αρχείο dataPreProcessing.ipynb και αναλύεται λίγο πιο αναλυτικά παρακάτω:

Αρχικά θα κάνουμε την διαδικασία του να δημιουργήσουμε το training/validation/test φακέλους με το παρακάτω format:

```
#Function to split folders of classes to Train , Validation and Test
# SEED is used for reproducible splitting
# IF root dir or/and destination dir are not defined then current directory of project will be used
```

```
#Format is like :
#INPUT:
# root_dir/
#    class1/
#        img1.jpg
#        img2.jpg
#        ...
#    class2/
#        img1.jpg
#        img2.jpg
#        ...
#    class3/
#        img1.jpg
#        img2.jpg
#        ...
#    ...
#
```

→

```
#OUTPUT:
# destination_dir/
#     train/
#         class1/
#             img1.jpg
#             ...
#         class2/
#             img1.jpg
#             ...
#     val/
#         class1/
#             img2.jpg
#             ...
#         class2/
#             imgb.jpg
#             ...
#     test/
#         class1/
#             img3.jpg
#             ...
#         class2/
#            imgc.jpg
#             ...
```

Έγραψα λοιπόν μια συνάρτηση για αυτήν την διαδικασία.

Όπως είπαμε και στην εισαγωγή η συνάρτηση που το κάνει αυτό βρίσκεται στο dataPreProcessing.ipynb και ορίζεται ως εξής:

```
def folder_splitting_dataset(classes_dir,val_ratio,test_ratio,SEED,root_dir=None,destination_dir=None):
```

Φυσικά δίνουμε ένα συγκεκριμένο seed για το ποιες εικόνες θα πάνε σε ποιόν φάκελο όπως και την αναλογία που θέλουμε για Training Validation και Testing.

Στην δικιά μου περίπτωση προτιμήθηκε η αναλογία 70/20/10 για train/val/test. (ακολουθώντας στο περίπου την ακολουθία Pareto).

Στην συνέχεια όπως είπαμε και παραπάνω θα φορτώσουμε αυτό το dataset σε NumPy Arrays με σκοπό να το αποθηκεύσουμε σε αυτή τη μορφή. Αυτό έγινε με την συνάρτηση:

```
def load_dataset(img_folder,grayScaled,normalized,IMG_HEIGHT,IMG_WIDTH):
```

Εδώ διαλέγουμε αν θέλουμε κανονικοποίηση, αν θέλουμε να κάνουμε όλες τις εικόνες grayscale καθώς και το μέγεθος στο οποίο θα έχουμε τις εικόνες.

```
img_data, class_name =load_dataset("C:\\Users\\Stelios\\Desktop\\FER_Custom_Dataset\\FER_Split\\val",True,False,64,64)
valX=np.array(img_data)
dictY={k: v for v, k in enumerate(np.unique(class_name))}
valY=  [dictY[class_name[i]] for i in range(len(class_name))]
np.save('valX.npy', valX)
np.save("valY.npy",valY)

img_data, class_name =load_dataset("C:\\Users\\Stelios\\Desktop\\FER_Custom_Dataset\\FER_Split\\train",True,False,64,64)

trainX=np.array(img_data)
dictY={k: v for v, k in enumerate(np.unique(class_name))}
trainY=  [dictY[class_name[i]] for i in range(len(class_name))]
np.save('trainX.npy', trainX)
np.save("trainY.npy",trainY)

img_data, class_name =load_dataset("C:\\Users\\Stelios\\Desktop\\FER_Custom_Dataset\\FER_Split\\test",True,False,64,64)
testX=np.array(img_data)
dictY={k: v for v, k in enumerate(np.unique(class_name))}
testY=  [dictY[class_name[i]] for i in range(len(class_name))]
np.save('testX.npy', testX)
np.save("testY.npy",testY)
```

Οπότε πλέον έχουμε NumPy arrays που περιέχουν το χωρισμένο dataset και τα labels του dataset για να χρησιμοποιήσουμε στο collab.

1. Απλό Τεχνητό Νευρωνικό Δίκτυο/Artificial Neural Network (ANN)

a) Στο Συγκεκριμένο μέρος της εργασίας καλούμαστε να δημιουργήσουμε fully connected νευρωνικά δίκτυα για το dataset που περιγράψαμε παραπάνω. Η αρχιτεκτονική που μας ζητείται περιέχει δύο κρυφά επίπεδα 256 και 128 νευρώνων αντίστοιχα.

Αρχικά θα φορτώσουμε το dataset από τα numpy arrays που δημιουργήσαμε παραπάνω.

```
[ ]  #load images as numpy arrays
     trainX=np.load('/content/drive/MyDrive/Colab Notebooks/trainX.npy')
     valX=np.load('/content/drive/MyDrive/Colab Notebooks/valX.npy')
     testX=np.load('/content/drive/MyDrive/Colab Notebooks/testX.npy')
     print("train images shape",trainX.shape)
     print("validation images shape",valX.shape)
     print("test images shape",testX.shape)
```

```
train images shape (30672, 64, 64)
validation images shape (8764, 64, 64)
test images shape (4383, 64, 64)
```

```
Train Images Per Class
Class 0 (Disappointed): 8913
Class 1 (Interested): 12047
Class 2 (Neutral): 9712
Validation Images Per Class
Class 0 (Disappointed): 2547
Class 1 (Interested): 3442
Class 2 (Neutral): 2775
Test Images Per Class
Class 0 (Disappointed): 1274
Class 1 (Interested): 1721
Class 2 (Neutral): 1388
```

Μπορούμε πλέον να δημιουργήσουμε το μοντέλο μας.

Για τον καθορισμό του του input layer πρέπει να δούμε πόσα pixel έχει η κάθε εικόνα μας καθώς κάθε input νευρώνας δέχεται 1 pixel.

Οι εικόνες μας είναι μεγέθους 64x64. Άρα θέλουμε 4096 το input layer.

Για το output layer έχουμε 3 κλάσεις άρα χρειαζόμαστε 3 νευρώνες και καθώς έχουμε ένα πρόβλημα με αποκλειστική ταξινόμηση (δηλαδή 1 εικόνα ανήκει μόνο σε 1 κλάση ,mutually exclusive) τότε είναι σημαντικό να χρησιμοποιήσουμε την softmax ώστε να έχουμε ένα τελικό διάνυσμα 3 θέσεων το άθροισμα των οποίων να ισούται με 1 και να αντιστοιχεί με την πιθανότητα της εικόνας να βρίσκεται σε 1 κατηγορία.

Το μοντέλο που περιγράψαμε λοιπόν είναι το εξής:

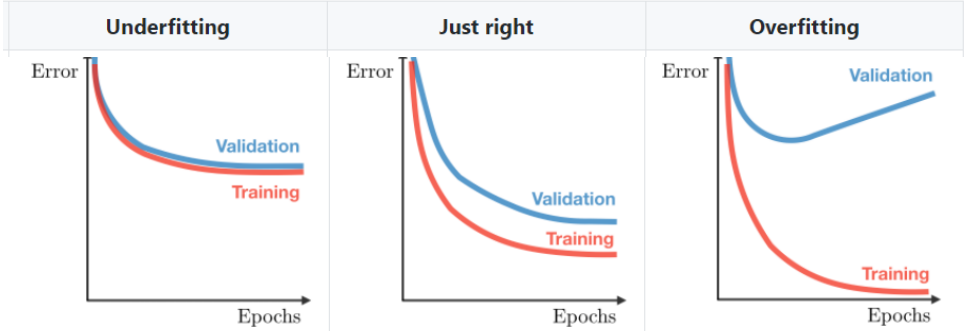
```
#First model building

model = models.Sequential(keras.layers.Flatten(input_shape = [64, 64]))
model.add(layers.Dense(256,activation='sigmoid' ,kernel_initializer="random_uniform"))
model.add(layers.Dense(128,activation='sigmoid', kernel_initializer="random_uniform"))
#we have 3 classes so last layer 3 neurons and softmax
model.add(layers.Dense(3,activation='softmax', kernel_initializer="random_uniform"))
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_8 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_24 (Dense)             (None, 256)               1048832
_________________________________________________________________
dense_25 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_26 (Dense)             (None, 3)                 387
=================================================================
Total params: 1,082,115
Trainable params: 1,082,115
Non-trainable params: 0
_________________________________________________________________
```

b) Έχοντας δημιουργήσει το μοντέλο μας θα περάσουμε στο μέρος της εκπαίδευσης και της εξαγωγής των ζητούμενων μετρικών.

Αρχικά για την εκπαίδευση πρέπει να ορίσουμε μερικές υπερπαραμέτρους.

- Θα ορίσουμε 500 εποχές εκπαίδευσης (μικρό δίκτυο προτιμήθηκε για οπτικούς λόγους στα γραφήματα που ζητήθηκαν στην εργασία να έχουμε πιο ολοκληρωμένα γραφήματα.
- Η μέθοδος εκπαίδευσης θα είναι το βασικό Stochastic Gradient Descend που είδαμε στο μάθημα με learning rate 1e-4. (έχω 500 εποχές προτίμησα ένα σχετικά μικρότερο lr,φυσικά υπάρχουν και άλλες επιλογές).
- Το batch size το ορίζουμε 32. είναι καλή πρακτική συνήθως να κρατάμε το batch size μικρό για καλυτερα generalization αποτελέσματα αποφεύγοντας πολύ sharp minima. (στο paper «On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima» υπάρχει εκτενείς περιγραφή του λόγου που αυτό συμβαίνει μαζί με δοκιμές που το επιβεβαιώνουν)
- Για να βρούμε τον βέλτιστο αριθμό εποχών θα θέλουμε να κρατήσουμε τα βάρη της εκπαίδευσης όπου το validation accuracy είναι μέγιστο ή το validation loss είναι ελάχιστο. Ο λόγος που ασχολούμαστε με το validation set είναι η ικανότητα να αναγνωρίσουμε αν έχουμε κάνει overfit/underfit το training dataset μας. Αρχικά δοκίμσα με το καλύτερο val_accuracy. (στην συνέχεια είδα καλύτερα αποτελέσματα με val_loss οπότε κρατήθηκε αυτό στα επόμενα ερωτήματα.



- Εισάγουμε λοιπόν callbacks και συγκεκριμένα checkpoint με σκοπό να αποθηκεύουμε τα βάρη από την καλύτερη εποχή με βάση την παραπάνω ανάλυση
- Επίσης για το μικρό imbalance που έχουμε όπως ανέφερα και πριν έκανα χρήση των class weights. Ουσιαστικά υπολογίζουμε και εισάγουμε βάρη για κάθε κλάση στην συνάρτηση κόστους μας έτσι ώστε το συνολικό σφάλμα που μπορούμε να έχουμε για κάθε κλάση να είναι ισοδύναμο . Για παράδειγμα αν είχαμε 2 κλάσεις θα υπολογίζαμε βάρη w0,w1 όπου υπολογίζονται : wj=n_samples / (n_classes * n_samples_j)και η συνάρτηση κόστους θα δώσει μεγαλύτερο ή μικρότερο σφάλμα για τα training samples της κάθε κλάσης ανάλογα με την κλάση που ανήκει.
- Η επιλογή του loss function είναι το “sparse_categorical_crossentropy” δηλαδή το loss function είναι το categorical cross entropy δίνοντας όμως τα labels μας με int τιμές και όχι σε one hot encoding. Αυτό θέλει προσοχή καθώς σε αντίθετη περίπτωση θα έπρεπε να χρησιμοποιήσω το “categorical_crossentropy” και αντί για class_weights να έδινα sample_weights στο keras.

Σε κώδικα τα παραπάνω:

```
[ ]  opt = keras.optimizers.SGD(1e-4)

     #to Keep best epoch waits
     filepath="/content/drive/MyDrive/Colab Notebooks/weights_best_1a.hdf5"
     checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')


     model.compile(loss = "sparse_categorical_crossentropy",
     optimizer =opt,
     metrics = ["accuracy"])


     history = model.fit(trainX,
                         trainY,
                         epochs = 500,
                         validation_data = (valX, valY),
                         shuffle=True,
                         batch_size=32,
                         class_weight=class_weight_dict,
                         callbacks=[checkpoint])
```

Μπορούμε πλέον να κάνουμε την εκπαίδευση , αλλά ας ορίσουμε πρώτα μια συνάρτηση για να εξάγουμε τις ζητούμενες μετρικές του ερωτήματος.

Στο ζητούμενο αυτό μας ζητείται να εξάγουμε μερικές μετρικές από την εκπαίδευση μας οι οποίες είναι:

Ορίζουμε για κάθε κλάση :

TN / True Negative: Δεν ανήκει στη κλάση και προβλέπουμε ότι δεν ανήκει στη κλάση

TP / True Positive: ανήκει στη κλάση και προβλέπουμε ότι ανήκει στη κλάση

FN / False Negative: Ανήκει στη κλάση αλλά προβλέπουμε ότι δεν ανήκει στη κλάση

FP / False Positive: Δεν ανήκει στη κλάση αλλά προβλέπουμε ότι ανήκει στη κλάση

Οπότε ορίζουμε για κάθε κλάση:

Precision = TP/(TP + FP), Ακρίβεια των θετικών προβλέψεων για μια κλάση

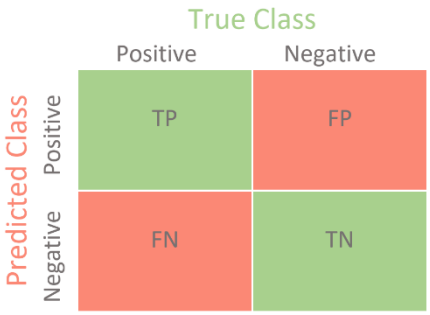
Recall = TP/(TP+FN), Το ποσοστό αυτών που ανήκουν στην κλάση και το μοντέλο ταξινόμησε αυτή τη κλάση

F1 Score = 2*(Recall * Precision) / (Recall + Precision) , Υβριδικό μετρικό που μας δίνει ένα weighted average μεταξύ του precision και του recall.

Φυσικά το accuracy είναι το ποσοστό των σωστών μας predictions. Το μετρικό αυτό έχει νόημα μόνο αν δεν έχουμε σημαντικό imbalance στις κλάσεις μας.

Η sk learn με την συνάρτηση classification report μας επιτρέπει να υπολογίσουμε όλα αυτά τα μετρικά.

Αναφορικά με το confusion matrix που ζητείται το υπολογίζουμε κανονικοποιημένο. Στην γενική του μορφή αποτελεί την παρακάτω οπτικοποίηση:



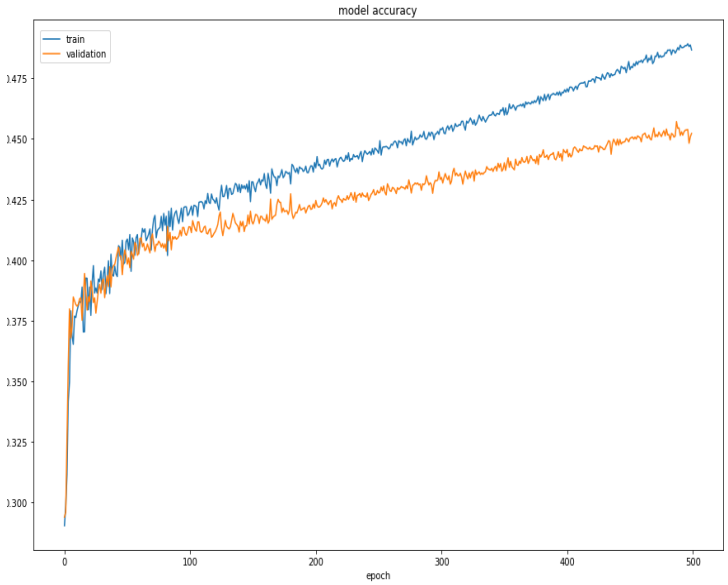
Οπότε έφτιαξα μια συνάρτηση που δίνουμε το μοντέλο που εκπαιδεύσαμε καθώς και ένα test dataset για τον υπολογισμό των μετρικών πέραν των γραφημάτων εκπαίδευσης και αυτό επιστρέφει όλες τις παραπάνω μετρικές:

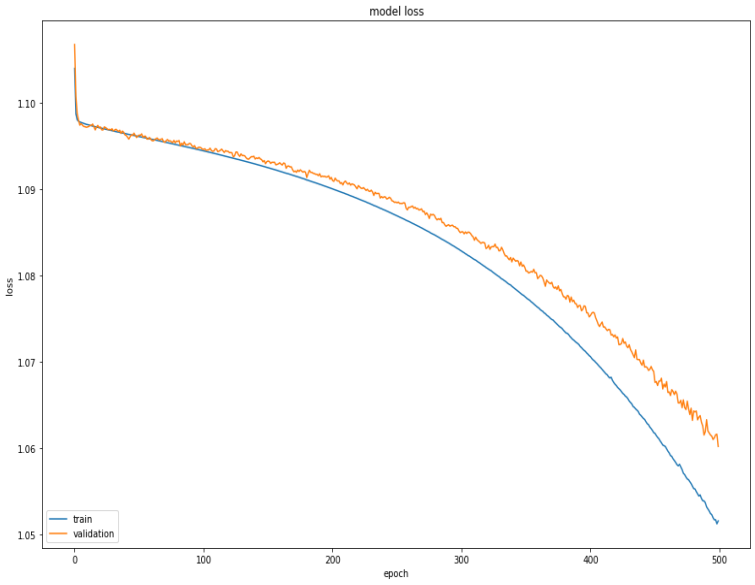
```
def evaluateAndGetMyMetrics(myModel,testDataset,testDatasetLabels):
```

Κάνοντας λοιπόν την εκπαίδευση για 500 εποχές καλούμε μετά για το μοντέλο και τα καλύτερα μας βάρη την συνάρτηση:

```
#get best weights for test evaluation
model.load_weights("/content/drive/MyDrive/Colab Notebooks/weights_best_1a.hdf5")
evaluateAndGetMyMetrics(model,testX,testY)
```

Και έχουμε τα παρακάτω αποτελέσματα:

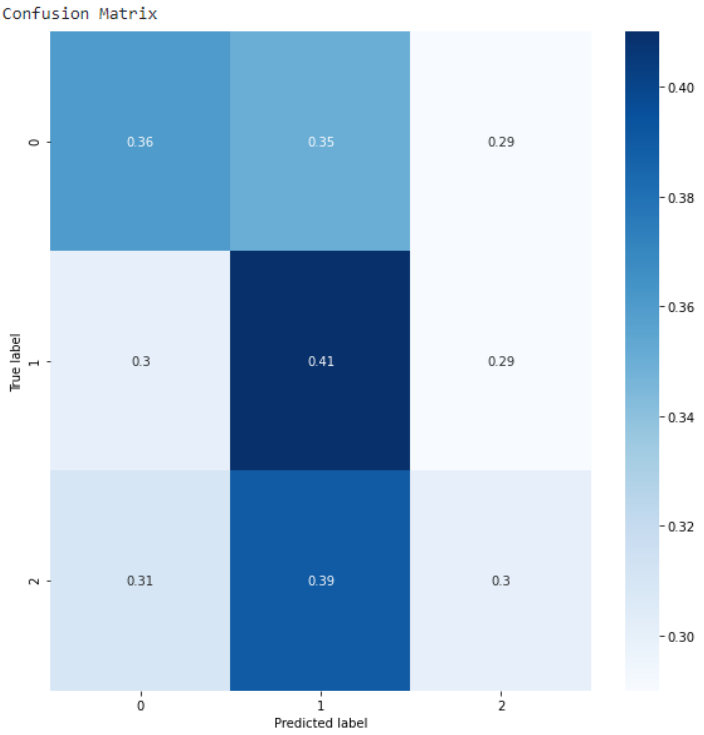




Και οι μετρικές από το test dataset

Test Metrics
Classification Report for Test Dataset

	precision	recall	f1-score	support
0	0.36	0.42	0.39	1274
1	0.52	0.59	0.55	1721
2	0.43	0.31	0.36	1388
accuracy			0.45	4383
macro avg	0.44	0.44	0.43	4383
weighted avg	0.45	0.45	0.44	4383



Από τα αποτελέσματα αρχικά παρατηρούμε ότι δεν έχουμε πολύ καλά αποτελέσματα στο test dataset. Αυτό ήταν αρκετά αναμενόμενο και λόγω της δυσκολίας του dataset καθώς και της απλότητας του δικτύου.

Αναφορικά με την εκπαίδευση, παρατηρούμε ότι δεν έχουμε σίγουρε overfitting. Φαίνεται να είμαστε αρκετά κοντά σε ένα καλό fit αλλά καθώς τα validation με το train loss curve είναι αρκετά κοντά και με τιμή πάνω από 1, είναι πιο πιθανό να έχουμε underfit στη συγκεκριμένη υλοποίηση. Ίσως μετά από 300 εποχές που αρχίζει να υπάρχει μια αυξανόμενη απόσταση μεταξύ train και validation ξεκινάει να κάνει overfit αλλά αν δούμε την κλίμακα της απόστασης αυτής είναι σίγουρο ότι δεν έχει γίνει ακόμα.

Για να αντιμετωπίσουμε το underfit ο καλύτερος τρόπος είναι να αυξήσουμε την πολυπλοκότητα του δικτύου μας.

c) Κάνουμε τις αλλαγές που ζητούνται στην εκφώνηση:

- Κανονικοποίηση δεδομένων στο [-1 ,1]

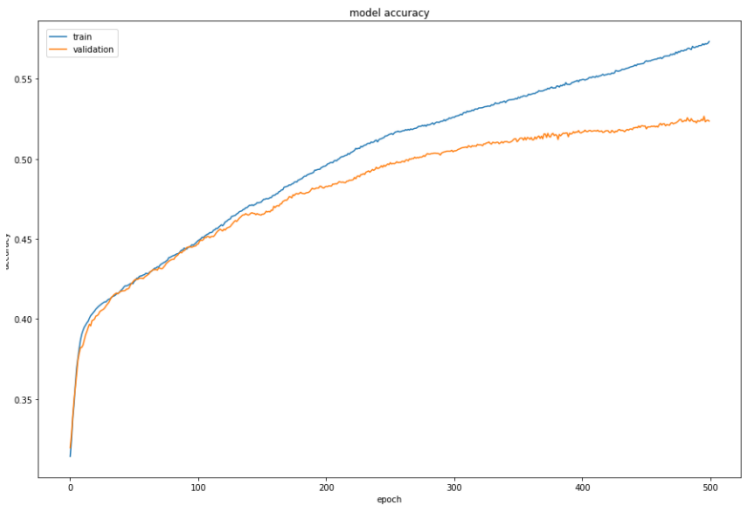
```
#normalize the data

oldRange = 255 - 0
newRange = 1 - (-1)

trainX_norm = ((trainX - 0) * newRange / oldRange) + (-1)
valX_norm= ((valX - 0) * newRange / oldRange) + (-1)
testX_norm=((testX - 0) * newRange / oldRange) + (-1)
```

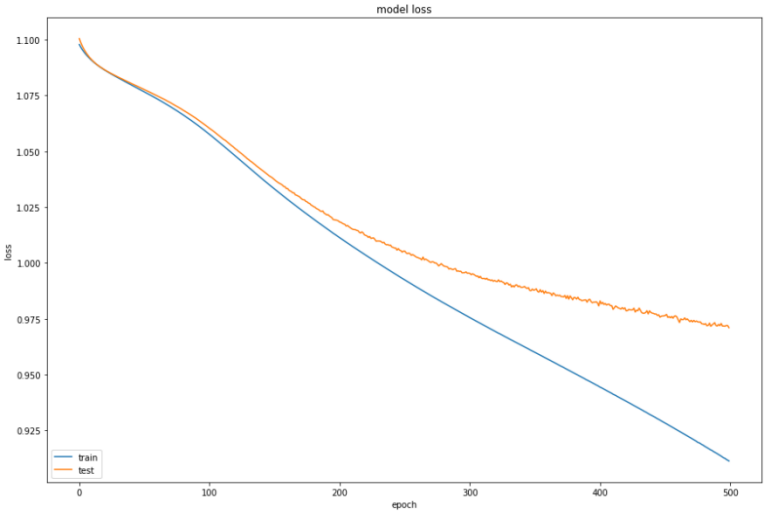
- Εισαγωγή της ReLu ως συνάρτηση ενεργοποίησης αντί της sigmoid.

```
#rebuild the model with ReLu

model = models.Sequential(keras.layers.Flatten(input_shape = [64, 64]))
model.add(layers.Dense(256,activation='relu' ,kernel_initializer="random_uniform"))
model.add(layers.Dense(128,activation='relu', kernel_initializer="random_uniform"))
#we have 3 classes so last layer 3 neurons and softmax
model.add(layers.Dense(3,activation='softmax', kernel_initializer="random_uniform"))
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_9 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_27 (Dense)             (None, 256)               1048832
_________________________________________________________________
dense_28 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_29 (Dense)             (None, 3)                 387
=================================================================
Total params: 1,082,115
Trainable params: 1,082,115
Non-trainable params: 0
_________________________________________________________________
```

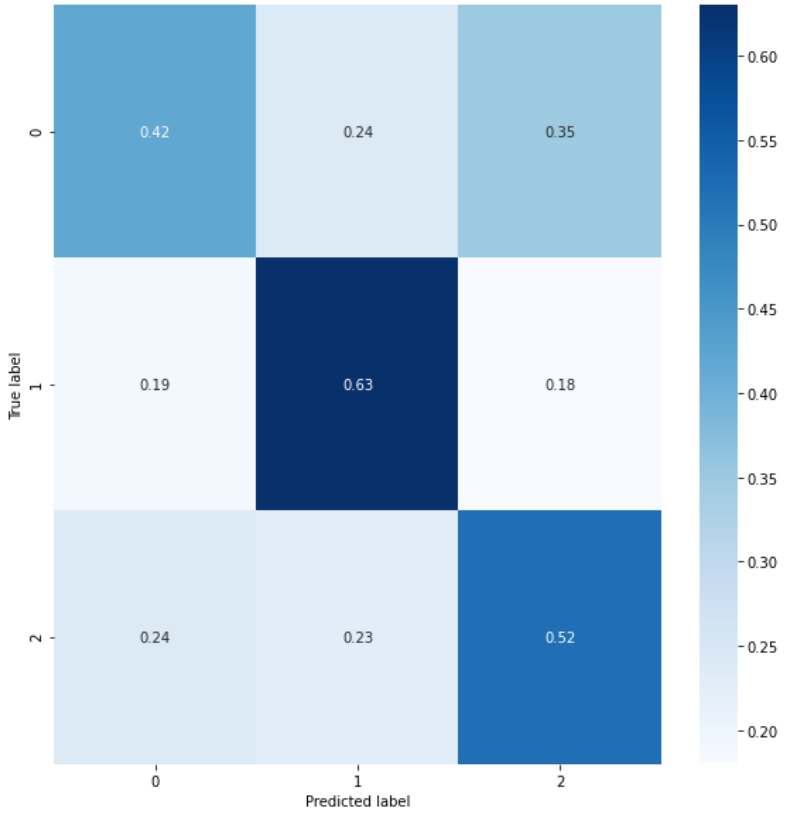
Σε αυτήν την υλοποίηση είχαμε τα παρακάτω αποτελέσματα





```
Test Metrics
Classification Report for Test Dataset
              precision    recall  f1-score   support

           0       0.44      0.42      0.43      1274
           1       0.63      0.63      0.63      1721
           2       0.49      0.52      0.51      1388

    accuracy                           0.53      4383
   macro avg       0.52      0.52      0.52      4383
weighted avg       0.53      0.53      0.53      4383
```

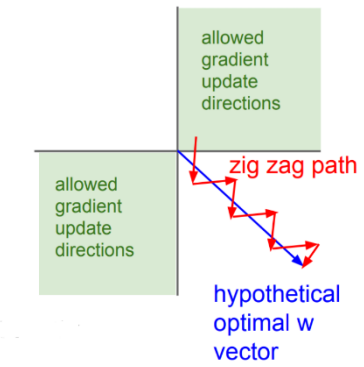
Confusion Matrix



Τα αποτελέσματα είναι σαφώς βελτιωμένα. Έχουμε βελτιωμένες όλες τις μετρικές μας. Επιπλέον το training ήταν πιο γρήγορο.

Τα αποτελέσματα είναι αναμενόμενα. Η κανονικοποίηση των δεδομένων μας γύρω από το 0 βοηθάει στην εκμάθηση με 2 τρόπους.

- Οι τιμές μας βρίσκονται πλέον μεταξύ του -1 και 1. Έχουμε λοιπόν και αρνητικές και θετικές τιμές ως inputs στα επόμενα layers. Αυτό κάνει το learning πιο flexible (σε συνδυασμό με την ReLu). Αυτό συμβαίνει λόγω της χρήσης του backpropagation. Αν είχαμε μόνο θετικά ή αρνητικά τότε τα βάρη μας θα κάναν updates για να καταλήξουν κάπου με έναν «zig-zag» τρόπο προς την τελική τους θέση κάνοντας την εκμάθηση πιο αργή και δύσκολη.



- Βρισκόμαστε πλέον καλύτερα μέσα στο εύρος των συναρτήσεων ενεργοποίησης έτσι οι πράξεις για τον υπολογισμό των εξόδων μας δεν βγάζουν πολύ μεγάλη έξοδο σε περίπτωση της relu) ή δεν έχουμε gradient saturation (στην περίπτωση της sigmoid). Έτσι έχουμε γρηγορότερο training

(Αν τα features μας δεν ήταν ήδη στο ίδιο range 0-255 τότε θα βοηθούσε επειδή θα είχαμε και ίδιο feature space για όλα τα features μας αλλά αυτό συμβαίνει ήδη)

Επίσης η ReLu είναι καλύτερη από την sigmoid για 2 βασικούς λόγος:

- Λύνει το πρόβλημα του vanishing gradient,όπου ουσιαστικά το gradient είναι αυξανόμενα μικρότερο (μάλιστα εκθετικά) καθώς προχωράμε πιο βαθιά μέσα στο δίκτυο(αυτό συμβαίνει και στην sigmoid και στην tanh). Αυτό σημαίνει ότι τα αρχικά layers του δικτύου σταματάνε να εκπαιδεύονται σωστά. Η ReLu έχει gradient 1 για είσοδο μεγαλύτερη του 0 οπότε το πρόβλημα των πολύ μικρών gradient λύνεται. (βέβαια μπορεί να έχουμε exploding gradients αν δεν το αντιμετωπίσουμε με άλλο τρόπο)
- Η εκπαίδευση είναι πιο γρήγορη καθώς η πράξη για την relu είναι πιο εύκολη (max(0,input)) σε σχέση με την εκθετική πράξη της sigmoid.

d) Σε αυτό το ερώτημα, μας ζητείται να κάνουμε χρήση του batch normalization. Ένα από τα προβλήματα που εμφανίζονται στα νευρωνικά δίκτυα είναι η αλλαγή της εισόδου των κρυφών επιπέδων δηλαδή η κατανομή αυτών των εισόδων. (Ονομάστηκε internal covariate shift από τους συγγραφείς του paper για το batch normalization). Αυτό οδηγεί και στην ανάγκη να μειώνουμε το learning rate στην διάρκεια της εκπαίδευσης. Για να λύσουμε αυτό το πρόβλημα θέλουμε να κανονικοποιήσουμε τις κατανομές των επιπέδων ενεργοποίησης κατά την εκπαίδευση έτσι ώστε να έχουν μέσο όρο 0 και απόκλιση 1. Αυτό το κάνουμε κάνοντας χρήση του Batch normalization. Για να γίνει αυτό υπολογίζουμε τον μέσο όρο και την απόκλιση παρτίδας:

$$\mu_\beta \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_\beta^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2$$

στη συνέχεια την κανονικοποιημένη ενεργοποίηση:

$$\hat{x_i} \leftarrow \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}}$$

και τέλος την αλλαγή κλίμακας και την μετατόπιση της:

$$y_i \leftarrow \gamma \hat{x_i} + \beta = BN_{\gamma,\beta}(x_i)$$

όπου x είναι οι ενεργοποιήσεις κάθε επιπέδου και γ,β είναι παράμετροι που πρέπει να μάθει το δίκτυο. Με βάση αυτά περιμένουμε και αύξηση των παραμέτρων του δικτύου. Επιπλέον λόγω των

παραπάνω υπολογισμών περιμένουμε και αύξηση του χρόνο εκπαίδευσης ανά εποχή . (αλλά θεωρητικά θα έχουμε πιο γρήγορο convergence)

Φυσικά εδώ αξίζει να σημειώσουμε 2 πράγματα. Αρχικά αν δεν έχουμε βαθύ δίκτυο, το input normalization μπορεί να είναι αρκετό για όλα τα layers. Επιπλέον υπάρχει ongoing έρευνα αναφορικά με το αν βοηθάει με αυτό το τρόπο και με ποιον τρόπο όντως βοηθάει στην εκπαίδευση (Αντίστοιχο paper από το MIT “How Does Batch Normalization Help Optimization”).

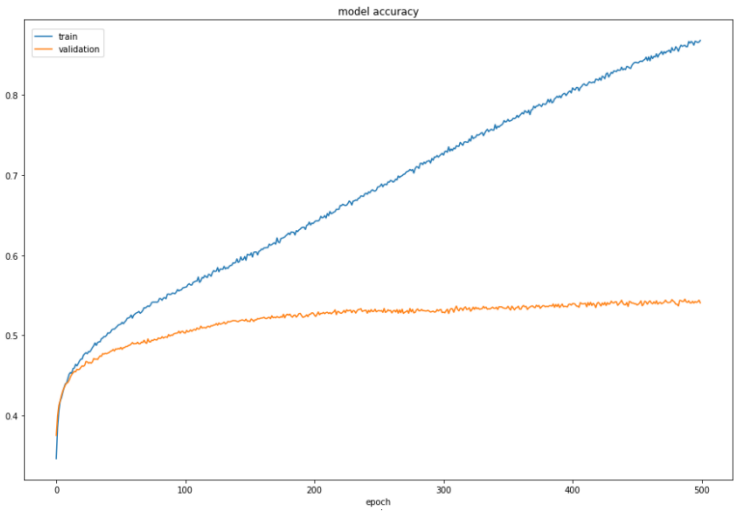
Περνώντας τώρα στην υλοποίηση θα δοκιμάσουμε να βάλουμε το batch norm πριν και μετά το activation function καθώς πρακτικά έχουν υπάρξει θετικά αποτελέσματα και για τα 2 (συνήθως προτιμάται το μετά)

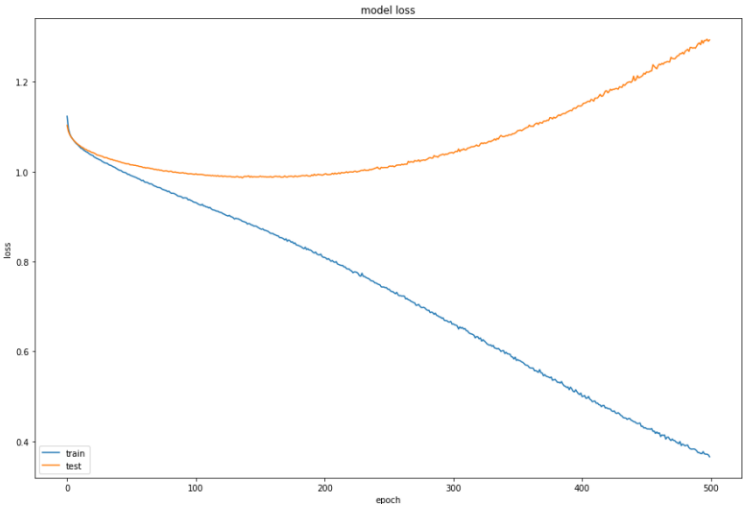
Έχουμε λοιπόν πρώτα το δίκτυο :

```
[ ]  model = models.Sequential(keras.layers.Flatten(input_shape = [64, 64]))
     model.add(layers.Dense(256,activation='relu' ,kernel_initializer="random_uniform"))
     model.add(layers.BatchNormalization())
     model.add(layers.Dense(128,activation='relu', kernel_initializer="random_uniform"))
     model.add(layers.BatchNormalization())
     #we have 3 classes so last layer 3 neurons and softmax
     model.add(layers.Dense(3,activation='softmax', kernel_initializer="random_uniform"))
     model.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_12 (Flatten)         (None, 4096)              0
_________________________________________________________________
dense_34 (Dense)             (None, 256)               1048832
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024
_________________________________________________________________
dense_35 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_3 (Batch (None, 128)               512
_________________________________________________________________
dense_36 (Dense)             (None, 3)                 387
=================================================================
Total params: 1,083,651
Trainable params: 1,082,883
Non-trainable params: 768
_________________________________________________________________
```

και τα παρακάτω αποτελέσματα:





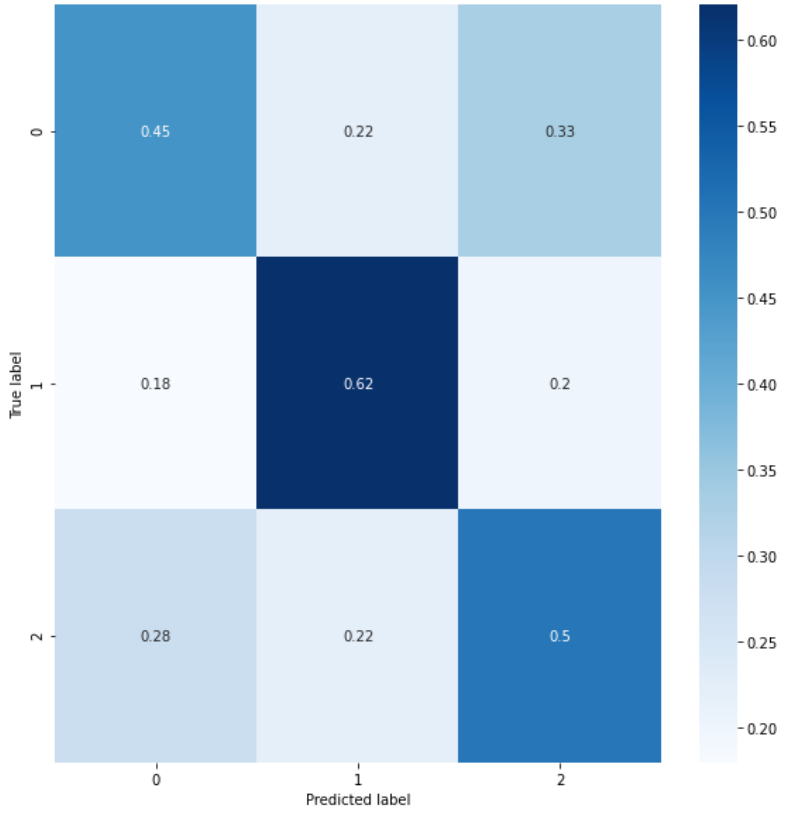
Παρατηρούμε ότι φτάνουμε πιο γρήγορα σε εποχές στα καλύτερα αποτελέσματα. Επιπλέον λόγω της επιτάχυνσης στην εκπαίδευση έχουμε πολύ νωρίτερα και overfit.

Στα test δεδομένα μας είχαμε αυτά τα αποτελέσματα.

Test Metrics
Classification Report for Test Dataset

	precision	recall	f1-score	support
0	0.45	0.45	0.45	1274
1	0.65	0.62	0.63	1721
2	0.47	0.50	0.49	1388
accuracy			0.53	4383
macro avg	0.52	0.52	0.52	4383
weighted avg	0.53	0.53	0.53	4383

Confusion Matrix

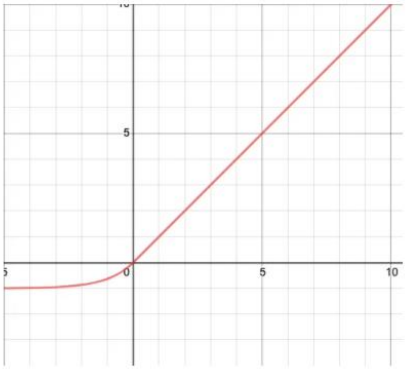


Δοκιμάζοντας και πριν το activation είχαμε πάρα πολύ κοντινά αποτελέσματα (δεν υπήρξε καμιά σοβαρή διαφορά, τα αποτελέσματα αναλυτικά βρίσκονται στο notebook και στην παρουσίαση).

e) Δοκιμάστηκαν συνολικά 4 αρχιτεκτονικές. Την προεπεξεργασία και το data augmentation το άφησα για τα cnn λόγω της εκμετάλλευσης του data locality (περισσότερα στο αντίστοιχο ερώτημα) Η λογική που ακολούθησα είναι η εξής: εφόσον το αρχικό δίκτυο έκανε underfit, θα δοκιμάσω μια πιο περίπλοκη αρχιτεκτονική με περισσότερα επίπεδα και περισσοτέρους νευρώνες ανά επίπεδο.

Η λογική της αρχιτεκτονικής ήταν να μην έχω "bottleneck" νευρώνων από επίπεδο σε επίπεδο αλλά να μειώνω σταδιακά μεχρι τους νευρώνες ανά επίπεδο.

Για να αποφύγω dying relu στο πιο βαθύ δίκτυο έκανα χρήση της ELU :



ELU

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

Επιπλέον άλλαξα τον optimizer από stochastic gradient descent σε Adam (Adaptive moment estimation). Εδώ κρατάμε decaying average από προηγούμενα gradients, ως εξής:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Αυτά τα momentums είναι biased προς το 0 οπότε υπολογίζεται το correction ως :

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Και ο κανόνας αναβάθμισης γίνεται :

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t.$$

Οι παράμετροι β1,β2 είναι υπερπαράμετροι.

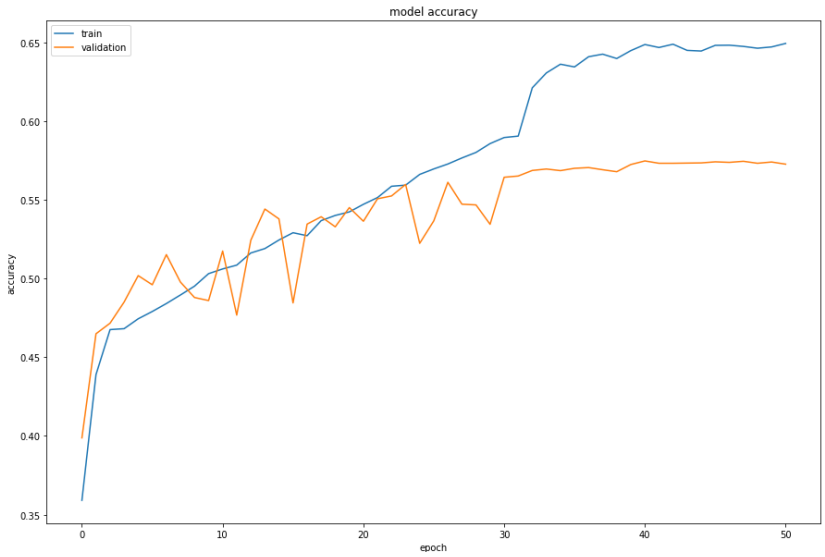
Επιπλέον παρατηρώντας στις πρώτες δοκιμαστικές εκδοχές overfit , προσέθεσα και dropout στα layers.
Το dropout ουσιαστικά τυχαία με μια πιθανότητα αγνοεί μερικά layer outputs . Αυτό εισάγει θόρυβο στην εκπαίδευση επιτρέποντας έτσι να μην «υπερβασίζεται» η εκπαίδευση σε συγκεκριμένους νευρώνες οδηγώντας σε καλύτερο generalization. Επιπλέον έχουμε ένα είδος προσομοίωσης διαφορετικών αρχιτεκτονικών νευρωνικών σε 1.
Τέλος κρατήθηκε το καλύτερο val loss αντί για το validation accuracy καθώς δοκιμάζοντας και τα 2 είχα καλύτερα γενικά αποτελέσματα με το validation loss, ενώ προστέθηκε και το callback για να μειώνει το learning rate όταν δεν έχω βελτίωσει του val_loss για 5 εποχές/

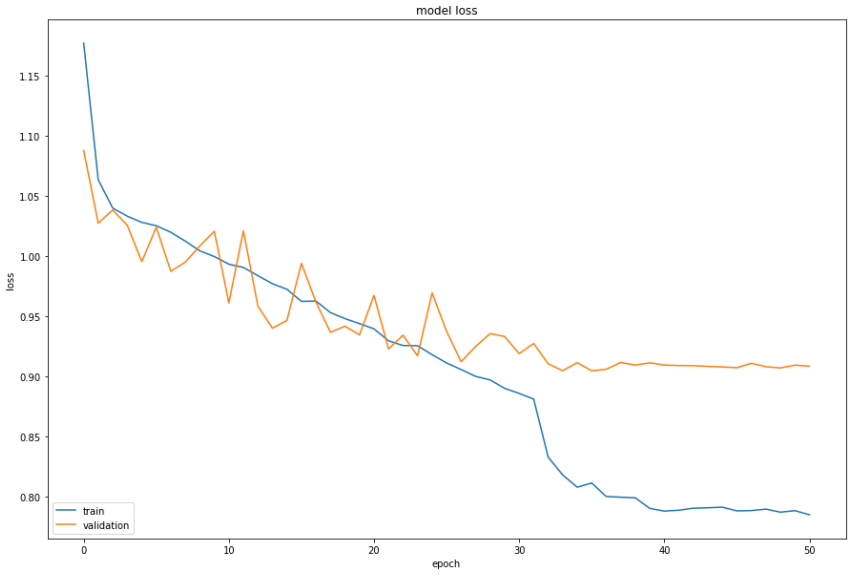
Αναλυτικά η κάθε δοκιμή με το κάθε αποτέλεσμα υπάρχει στο notebook. Η καλύτερη αρχιτεκτονική ήταν η εξης:

```
model = models.Sequential(keras.layers.Flatten(input_shape = [64, 64]))


model.add(layers.Dense(2048))
model.add(layers.BatchNormalization())
model.add(layers.Activation('elu'))
model.add(layers.Dropout(0.5))


model.add(layers.Dense(1024))
model.add(layers.BatchNormalization())
model.add(layers.Activation('elu'))
model.add(layers.Dropout(0.5))


model.add(layers.Dense(512))
model.add(layers.BatchNormalization())
model.add(layers.Activation('elu'))
model.add(layers.Dropout(0.5))


model.add(layers.Dense(512))
model.add(layers.BatchNormalization())
model.add(layers.Activation('elu'))
model.add(layers.Dropout(0.5))


model.add(layers.Dense(256))
model.add(layers.BatchNormalization())
model.add(layers.Activation('elu'))
model.add(layers.Dropout(0.5))


model.add(layers.Dense(256))
model.add(layers.BatchNormalization())
model.add(layers.Activation('elu'))
model.add(layers.Dropout(0.5))

#we have 3 classes so last layer 3 neurons and softmax
model.add(layers.Dense(3,activation='softmax'))
model.summary()
```

```
=================================
Total params: 11,492,611
Trainable params: 11,483,395
Non-trainable params: 9,216
```

έφτασε στα παρακάτω αποτελέσματα:

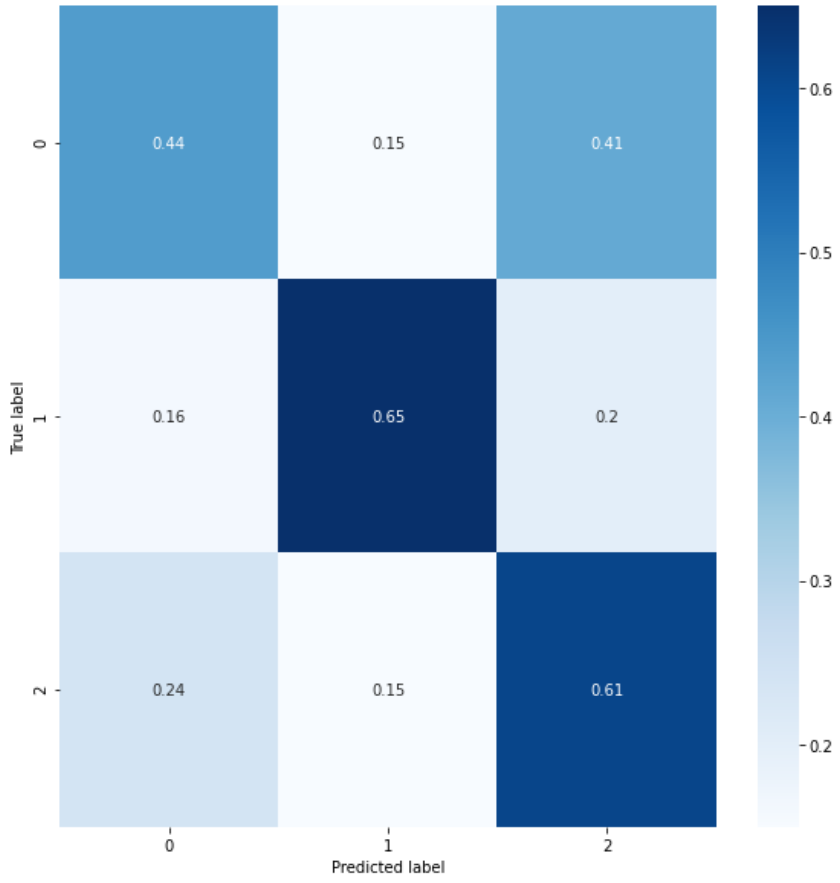




Test Metrics
Classification Report for Test Dataset

	precision	recall	f1-score	support
0	0.48	0.44	0.46	1274
1	0.74	0.65	0.69	1721
2	0.49	0.61	0.55	1388
accuracy			0.57	4383
macro avg	0.57	0.56	0.56	4383
weighted avg	0.59	0.57	0.58	4383

Confusion Matrix



2. Συνελικτικά Νευρωνικά Δίκτυα/Convolutional Neural Networks (CNN)

a) Περνώντας τώρα στα CNN θα κάνουμε την ίδια αρχική επεξεργασία των δεδομένων με μόνη αλλαγή το shape της εικόνας πλέον πρέπει να είναι (64,64,1).
Δημιουργούμε το δίκτυο που ζητείται:

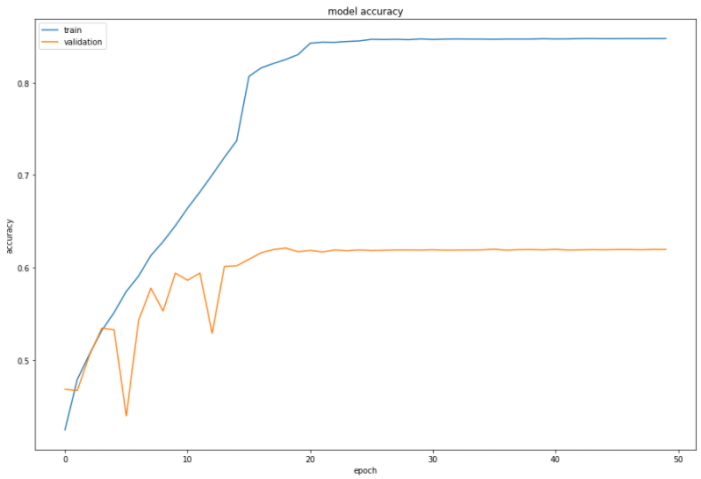
```
model = models.Sequential()
model.add(layers.Conv2D(64, (3, 3), input_shape=(64,64,1),padding='same'))
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D())
model.add(layers.Conv2D(32, (3, 3),padding='same'))
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D())
model.add(layers.Flatten())
# Fully connected layer
model.add(layers.Dense(128))
model.add(layers.Activation('relu'))
#add last layer for 3 classes
model.add(layers.Dense(3))
model.add(layers.Activation('softmax'))
model.summary()
```

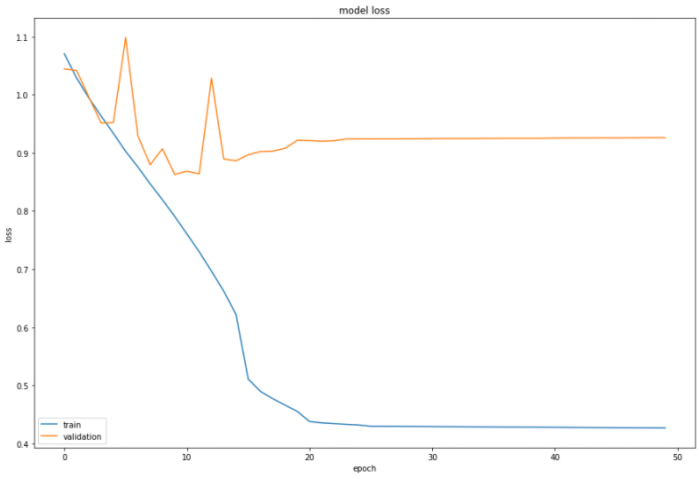
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 64)        640
_________________________________________________________________
activation (Activation)      (None, 64, 64, 64)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        18464
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 128)               1048704
_________________________________________________________________
activation_2 (Activation)    (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 387
_________________________________________________________________
activation_3 (Activation)    (None, 3)                 0
=================================================================
Total params: 1,068,195
Trainable params: 1,068,195
Non-trainable params: 0
_________________________________________________________________
```

Με τις παρακάτω παραμέτρους εκπαίδευσης:

```
opt = keras.optimizers.SGD(1e-2)

#to Keep best epoch waits
filepath="/content/drive/MyDrive/Colab Notebooks/weights_best_2a.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
                              patience=5, min_lr=1e-5,verbose=1)

model.compile(loss = "sparse_categorical_crossentropy",
optimizer =opt,
metrics = ["accuracy"])


history = model.fit(trainX_norm,
                    trainY,
                    epochs = 50,
                    validation_data = (valX_norm, valY),
                    shuffle=True,
                    batch_size=32,
                    class_weight=class_weight_dict,
                    callbacks=[checkpoint,reduce_lr])
```

b) Έχουμε τα παρακάτω αποτελέσματα:

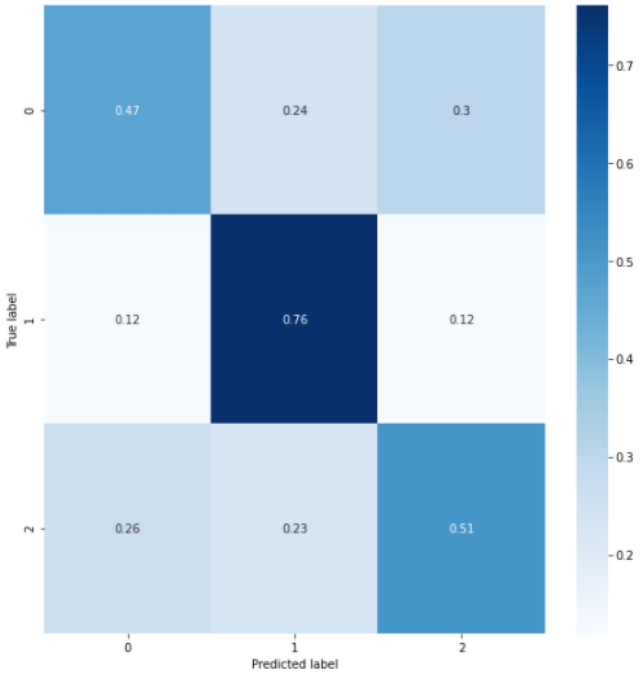




Test Metrics
Classification Report for Test Dataset

	precision	recall	f1-score	support
0	0.51	0.47	0.49	1274
1	0.68	0.76	0.72	1721
2	0.55	0.51	0.53	1388
accuracy			0.60	4383
macro avg	0.58	0.58	0.58	4383
weighted avg	0.59	0.60	0.59	4383

Confusion Matrix



Αρχικά παρατηρούμε ότι τα αποτελέσματα μας είναι καλύτερα από όλα τα ANN. Αυτό είναι αναμενόμενο και θα αναλυθεί περισσότερο στην αντίστοιχη ερώτηση θεωρίας. Επιπλέον βλέπουμε ένα overfit να συμβαίνει. Κάποιοι συνήθεις τρόποι για να το αντιμετωπίσουμε θα μπορούσαμε είτε να εισάγουμε dropout ή να κάνουμε data augmentations (θα γίνουν και τα 2 στις δοκιμαστικές υλοποιήσεις).

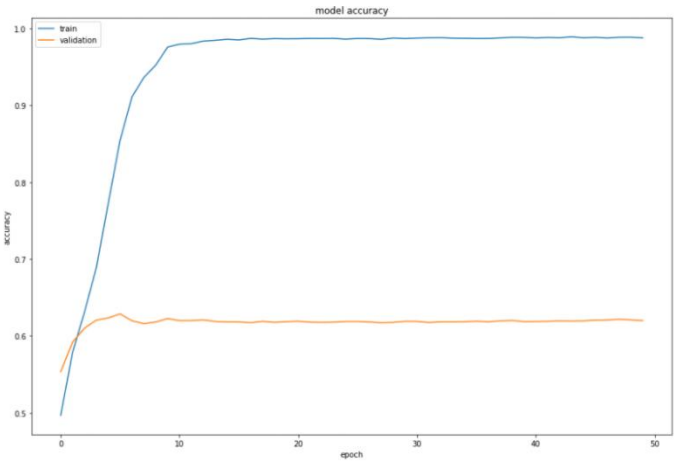
c) Εδώ δοκιμάστηκαν διαφορετικές αρχιτεκτονικές και συναρτήσεις ενεργοποίησης. Η λογική που ακολούθησα στις αρχιτεκτονικές μου ήταν η εξής . Τα αρχικά filters αναγνωρίζουν γεωμετρικά χαρακτηριστικά (low level- ακμές, ευθείες ,γωνίες) ενώ τα βαθύτερα layer ανώτερα χαρακτηριστικά δηλαδή συνδυασμούς των low level . Επειδή τα βασικά low level χαρακτηριστικά είναι λιγότερα από τους δυνατούς συνδυασμούς αυτών για την δημιουργία high level χαρακτηριστικών ,προτίμησα ο αριθμός των filter να αυξάνεται καθώς πάμε βαθύτερα στο δίκτυο. Άλλες επιλογές είχαν παρόμοια λογική με την περιγραφή που έκανα και στο ANN (όπως πχ το dropout).
Η δομή που κατέληξα είναι η εξής:

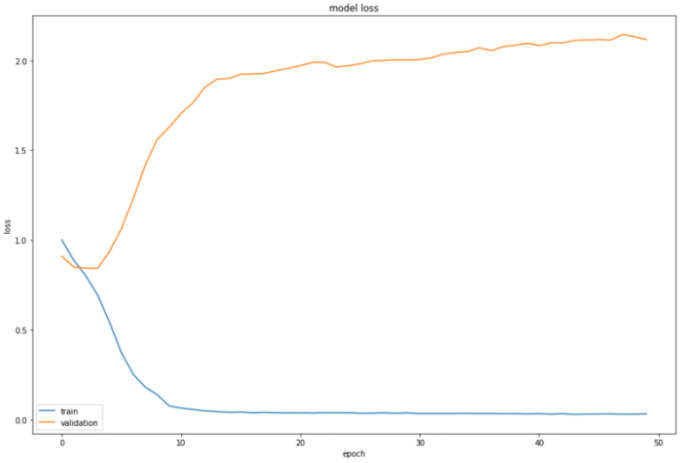
```
[ ]  model = models.Sequential()

     model.add(layers.Conv2D(16, (5, 5), input_shape=(64,64,1),padding='same'))
     model.add(layers.Activation('swish'))

     model.add(layers.Conv2D(32, (3, 3),padding='same'))
     model.add(layers.Activation('swish'))

     model.add(layers.Conv2D(32, (3, 3),padding='same'))
     model.add(layers.Activation('swish'))
     model.add(layers.MaxPooling2D(pool_size=(2,2)))

     model.add(layers.Conv2D(64, (3, 3),padding='same'))
     model.add(layers.Activation('swish'))

     model.add(layers.Conv2D(64, (3, 3),padding='same'))
     model.add(layers.Activation('swish'))
     model.add(layers.MaxPooling2D(pool_size=(2,2)))

     model.add(layers.Flatten())

     # Fully connected layer
     model.add(layers.Dense(256))
     model.add(layers.Activation('swish'))
     model.add(layers.Dropout(0.3))

     model.add(layers.Dense(128))
     model.add(layers.Activation('swish'))
     model.add(layers.Dropout(0.3))

     #add last layer for 3 classes
     model.add(layers.Dense(3))
     model.add(layers.Activation('softmax'))
     model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 64, 64, 16)	416
activation_4 (Activation)	(None, 64, 64, 16)	0
conv2d_3 (Conv2D)	(None, 64, 64, 32)	4640
activation_5 (Activation)	(None, 64, 64, 32)	0
conv2d_4 (Conv2D)	(None, 64, 64, 32)	9248
activation_6 (Activation)	(None, 64, 64, 32)	0
max_pooling2d_2 (MaxPooling2	(None, 32, 32, 32)	0
conv2d_5 (Conv2D)	(None, 32, 32, 64)	18496
activation_7 (Activation)	(None, 32, 32, 64)	0
conv2d_6 (Conv2D)	(None, 32, 32, 64)	36928
activation_8 (Activation)	(None, 32, 32, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 16, 16, 64)	0
flatten_1 (Flatten)	(None, 16384)	0
dense_2 (Dense)	(None, 256)	4194560
activation_9 (Activation)	(None, 256)	0
dropout (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 128)	32896
activation_10 (Activation)	(None, 128)	0
dropout_1 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 3)	387
activation_11 (Activation)	(None, 3)	0

Total params: 4,297,571
Trainable params: 4,297,571
Non-trainable params: 0

Τα αποτελέσματα είναι τα παρακάτω:

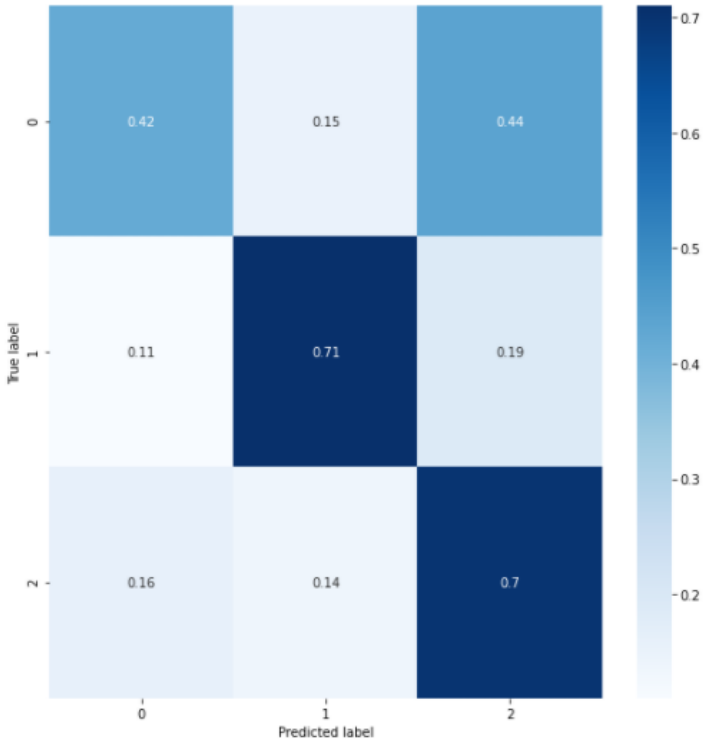




Test Metrics
Classification Report for Test Dataset

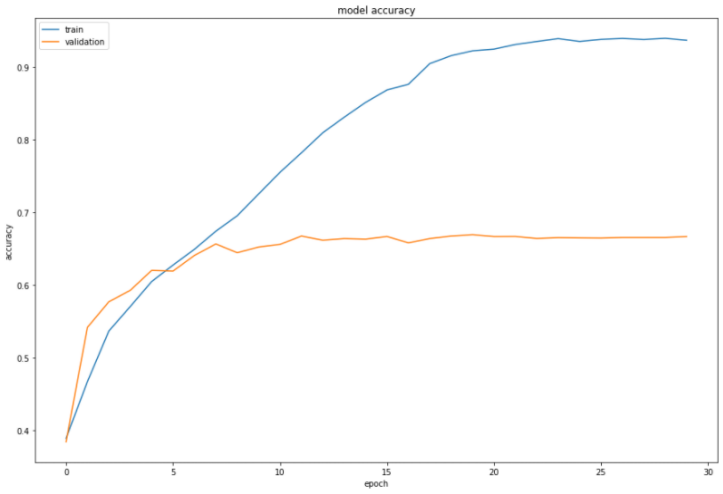
	precision	recall	f1-score	support
0	0.57	0.42	0.48	1274
1	0.76	0.71	0.73	1721
2	0.52	0.70	0.60	1388
accuracy			0.62	4383
macro avg	0.62	0.61	0.60	4383
weighted avg	0.63	0.62	0.62	4383

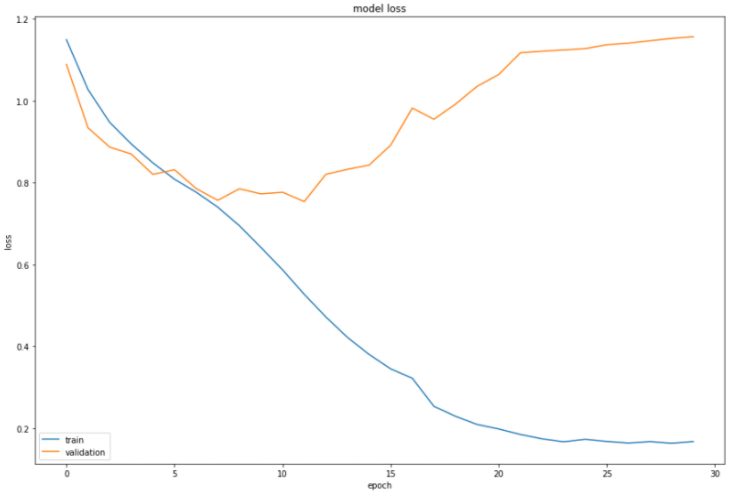
Confusion Matrix



Έχουμε βελτίωση στα αποτελέσματα αλλά πάλι αντιμετωπίζουμε overfit.

d) Εισάγοντας batch normalization (πριν το activation) όπως περιγράψαμε στο αντίστοιχο ερώτημα του ANN, τα αποτελέσματα ήταν τα παρακάτω

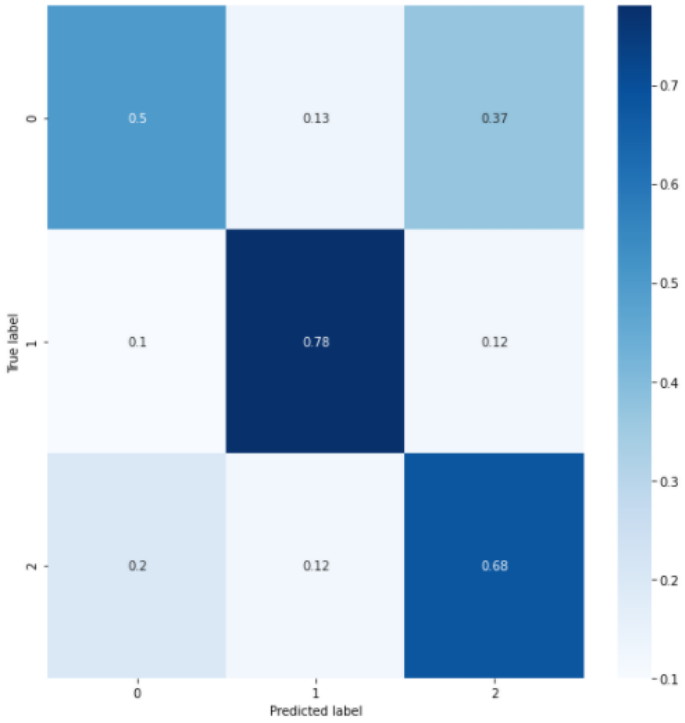




Test Metrics
Classification Report for Test Dataset

	precision	recall	f1-score	support
0	0.58	0.50	0.53	1274
1	0.80	0.78	0.79	1721
2	0.58	0.68	0.63	1388
accuracy			0.66	4383
macro avg	0.65	0.65	0.65	4383
weighted avg	0.67	0.66	0.66	4383

Confusion Matrix



Τα αποτελέσματα είναι πολύ βελτιωμένα. Σε αυτήν την περίπτωση το batch normalization βοήθησε και στη μείωση του overfit. Αυτό σύμφωνα με τον Ian goodfellow συμβαίνει καθώς με το να αλλάζουμε την κατανομή στην είσοδο κάθε επιπέδου κατά κάποιον τρόπο εισάγουμε «θόρυβο» στην εκπαίδευση μας.

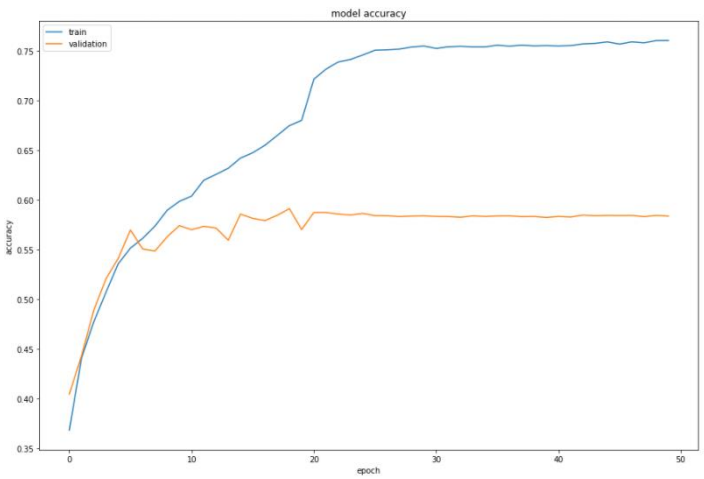
e) Στην συνέχεια μας ζητείται να κάνουμε μια αρχιτεκτονική χωρίς fully connected layer. Αυτό είναι δυνατόν να δουλέψει μόνο υπο προϋποθέσεις. Δηλαδή ο λόγος που χρειαζόμαστε fully connected layer είναι κυρίως για να καταλήξουμε σε νευρώνες που αντιστοιχούν στον αριθμό των κλάσεων μας καθώς και για να για να έχουμε global συνδυασμούς των local features που έχουμε κάνει extract από τα convolutions που έχουμε κάνει. Δηλαδή μπορούμε να σκεφτούμε το CNN μέρος του δικτύου ότι μας δίνει representations του input image και το fully connected μέρος χρησιμοποιείται για το classification .
Μπορεί λοιπόν να αντικατασταθεί θεωρητικά με άλλου είδους τακτικές για classification.
Για παράδειγμα υπάρχουν αρχιτεκτονικές που μετά το CNN γίνεται χρήση SVM.
Επιπλέον υπάρχουν τα Fully Convolutional Networks όπου δεν έχουμε fully connected layers. Αυτήν την λογική ακολούθησα στο δικό μου δίκτυο.
Πιο συγκεκριμένα:
Θα κάνουμε χρήση των 1x1 convolutions . και του Global average pooling με αυτόν τον τρόπο:

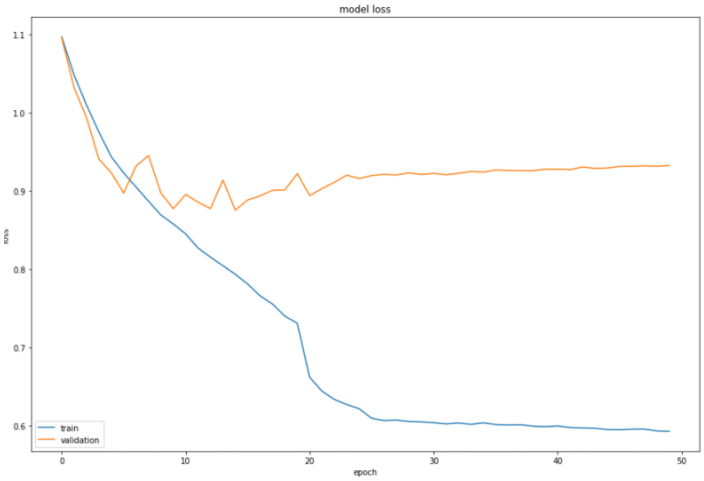
```
#Replace the Fully conected layer with 1x1 Convolution and Global average pooling
model.add(layers.Conv2D(3, (1, 1),padding='same'))
model.add(layers.GlobalMaxPooling2D())
model.add(layers.BatchNormalization())
model.add(layers.Activation('softmax'))
```

Έτσι λοιπόν το αποτέλεσμα που έχουμε στα vectors μας είναι το εξής. Πάμε από χώρο 16x16x32 που είμασταν σε χώρο 16x16x3 λόγω των 3 filters του 1x1 convolution. Στη συνέχεια με το global average pooling έχουμε πλέον 3 τιμές που αποτελούν την τιμή για κάθε κλάση. Ουσιαστικά με το 1x1 convolution κάναμε ένα feature map των features μας για κάθε κλάση και από αυτό με το global average pooling καταλήγουμε σε 1 τιμή για κάθε κλάση καθώς υπολογίζει το average για κάθε feature map από τα 3 που έχουμε.

```
batch_normalization_22 (Batc (None, 32, 32, 32)        128
_________________________________________________________________
activation_22 (Activation)   (None, 32, 32, 32)        0
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 16, 16, 3)         99
_________________________________________________________________
global_max_pooling2d_3 (Glob (None, 3)                 0
_________________________________________________________________
batch_normalization_23 (Batc (None, 3)                 12
_________________________________________________________________
activation_23 (Activation)   (None, 3)                 0
=================================================================
```

Κάνοντας την εκπαίδευση έχουμε παρόμοια αποτελέσματα με πριν:

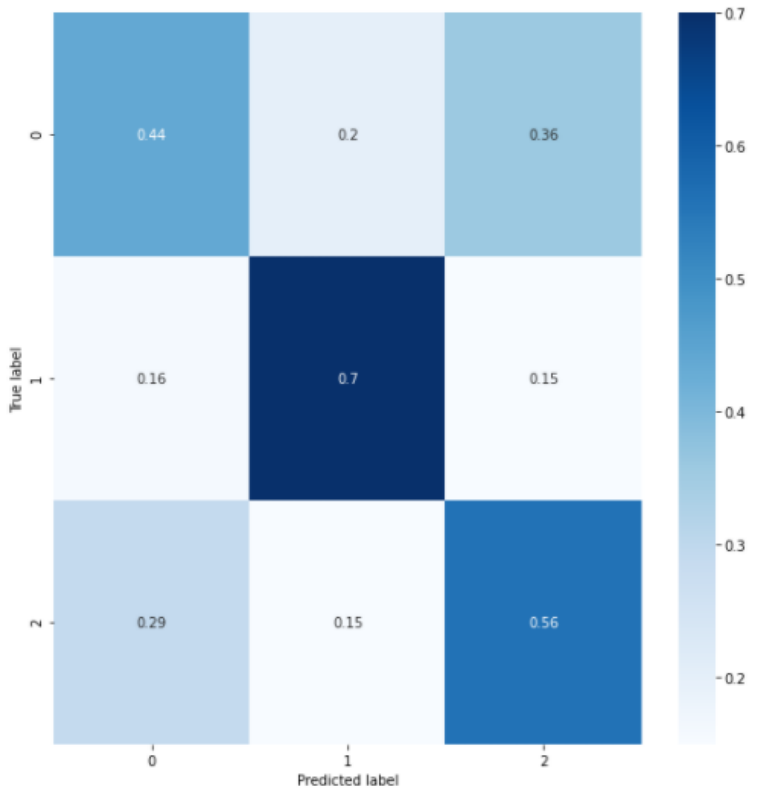




Test Metrics
Classification Report for Test Dataset

	precision	recall	f1-score	support
0	0.45	0.44	0.44	1274
1	0.72	0.70	0.71	1721
2	0.52	0.56	0.54	1388
accuracy			0.58	4383
macro avg	0.57	0.57	0.57	4383
weighted avg	0.58	0.58	0.58	4383

Confusion Matrix



Αν και τα αποτελέσματα προέκυψαν ελαφρώς χειρότερα, ίσως με κάποια παραπάνω παραμετροποίηση στα 1x1 convolutional layers μπορούμε να φτάσουμε σε αντίστοιχα αποτελέσματα. Παρόλα αυτά το δίκτυο δούλεψε κανονικά και μάλιστα εκπαιδεύτηκε πιο γρήγορα.

Στο κομμάτι που μας ζητείται να χρησιμοποιήσουμε ένα cnn χωρίς pooling δίκτυα (θεωρώ ότι εννοεί στο βασικό και όχι σε αυτό χωρίς dense layers γιατί εκεί χρειαζόμαστε κάποιο global pooling layer)

Οπότε στην υλοποίηση του d) αφαίρεσα όλα τα pooling layers:

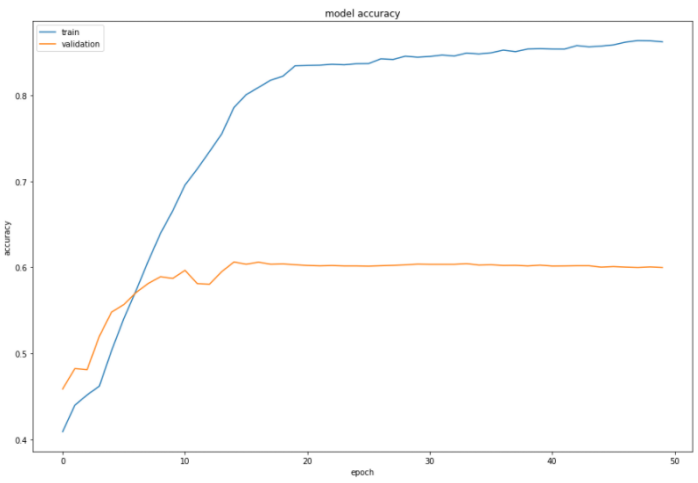
```
model = models.Sequential()

model.add(layers.Conv2D(8, (5, 5), input_shape=(64,64,1)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('swish'))

model.add(layers.Conv2D(16, (3, 3)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('swish'))

model.add(layers.Conv2D(16, (3, 3)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('swish'))

model.add(layers.Conv2D(32, (3, 3)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('swish'))

model.add(layers.Conv2D(32, (3, 3)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('swish'))

model.add(layers.Flatten())

# Fully connected layer
model.add(layers.Dense(256))
model.add(layers.Activation('swish'))
model.add(layers.Dropout(0.3))

model.add(layers.Dense(128))
model.add(layers.Activation('swish'))
model.add(layers.Dropout(0.3))

#add last layer for 3 classes
model.add(layers.Dense(3))
model.add(layers.Activation('softmax'))
model.summary()
```

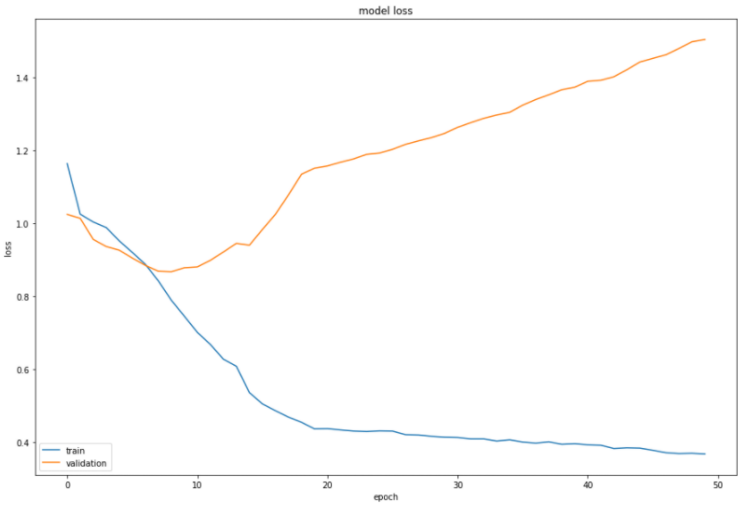
Εδώ παρατηρούμε μια πολύ λογική μεγάλη αύξηση των παραμέτρων του δικτύου:

```
Total params: 22,202,707
Trainable params: 22,202,499
Non-trainable params: 208
```

Αυτό είναι αναμενόμενο καθώς το pooling λειτουργεί σαν επίπεδο υποδειγματοληψίας μειώνοντας το output συνολικό shape του επιπέδου.

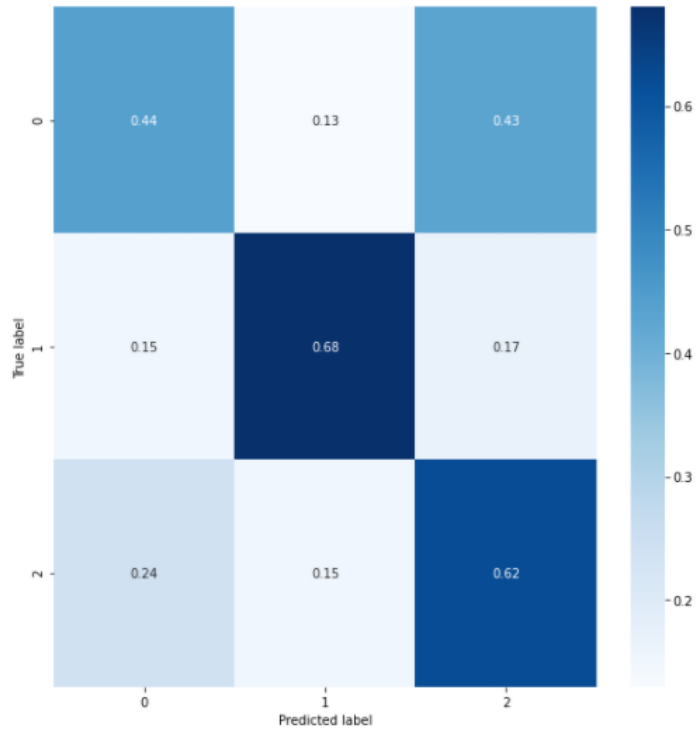
Κάνοντας εκπαίδευση σε αυτό το δίκτυο :





```
Test Metrics
Classification Report for Test Dataset
              precision    recall  f1-score   support

           0       0.49      0.44      0.46      1274
           1       0.76      0.68      0.72      1721
           2       0.51      0.62      0.56      1388

    accuracy                           0.59      4383
   macro avg       0.58      0.58      0.58      4383
weighted avg       0.60      0.59      0.59      4383
```

Confusion Matrix



Τα αποτελέσματα είναι σαφώς χειρότερα σε σχέση με το αντίστοιχο δίκτυο με τα pooling layers. Αυτό είναι αναμενόμενο καθώς βοηθάνε στην μείωση του variance στο δίκτυο καθώς και στη μείωση του overfitting.

Μια εναλλακτική θα μπορούσε να είναι η χρήση dilated convolution αλλά στα πλαίσια της εργασίας δεν δοκιμάστηκε.

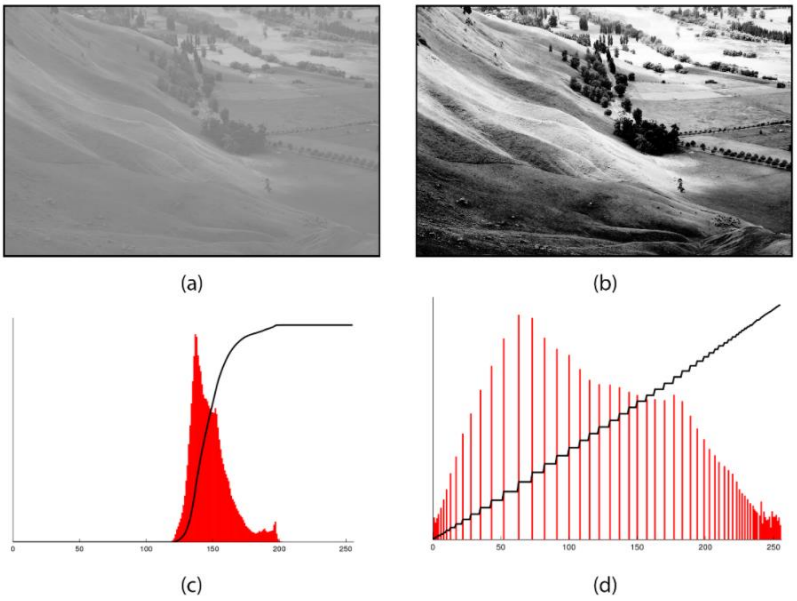
2.1 Η καλύτερη Συνολική Υλοποίηση χρησιμοποιώντας Data preprocessing και Data augmentations

Έχοντας δοκιμάσει πολλές υλοποιήσεις σε αρχιτεκτονικές και optimizers και υπερπαραμέτρους είδα ότι τα καλύτερα νούμερα τα είχα στην αρχιτεκτονική που περιγράψαμε στο ερώτημα d).

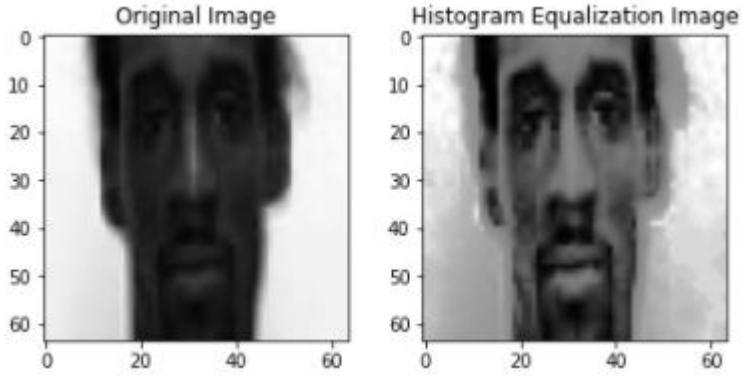
Για να μπορέσω να αυξήσω ακόμα παραπάνω τα αποτελέσματα αποφάσισα να ασχοληθώ πλέον με το dataset. Πιο συγκεκριμένα παρατήρησα ότι πολλές grayscale εικόνες (ειδικά αυτές που μετατράπηκαν από έγχρωμες) δεν φαινόντουσαν σωστά. Ήταν πολύ σκοτεινές ή φωτεινές. Αποφάσισα λοιπόν να κάνω μια προ επεξεργασία με σκοπό να κάνω μετά και data augmentations

που θα οδηγήσουν σε καλύτερο Generalization . Η τακτική που ακολούθησα ήταν το Histogram Equalization

Αυτό που κάνουμε σε αυτήν την διαδικασία είναι να αλλάξουμε το contrast της εικόνας με βάση το ιστόγραμμα της εικόνας.

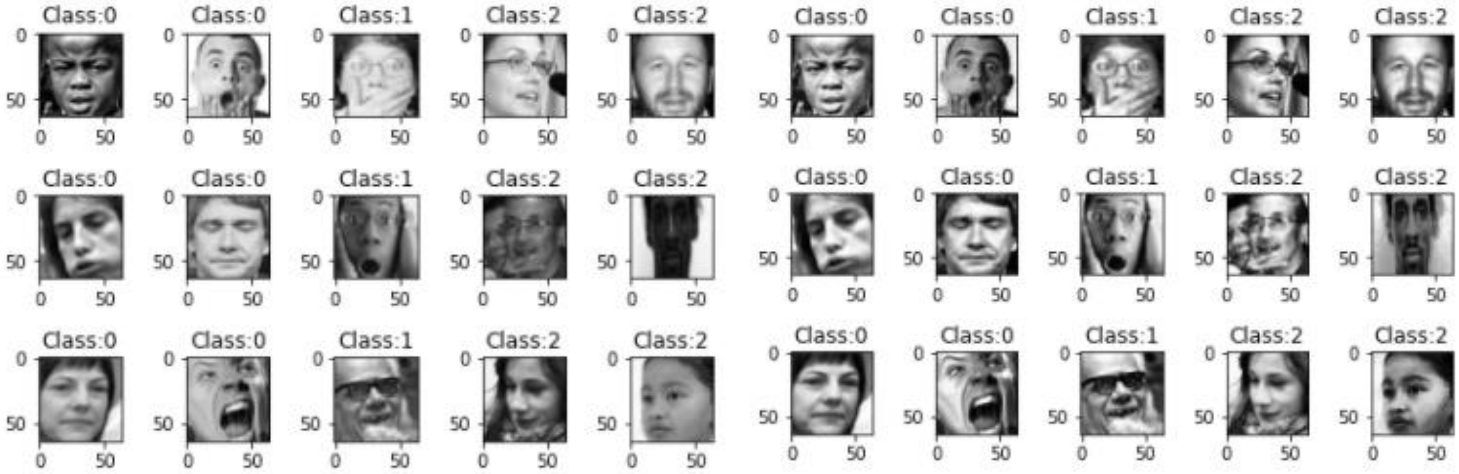


Αυτό έχει ως αποτέλεσμα να τονιστούν χαρακτηριστικά του προσώπου. Δοκιμάζοντας το στο dataset είχαμε βελτίωση εικόνων όπως πχ:



Στην συνέχεια αφού κάνουμε αυτό το μετασχηματισμό σε όλο το dataset

Πριν Μετά



Έφτιαξα μια συνάρτηση για αυτήν την διαδικασία σε όλο το dataset

```
[ ]  def datasetHistogramEqualization(dataset):
       for i in range(dataset.shape[0]):
         dataset[i] = exposure.equalize_hist(dataset[i])
       return dataset
```

```
[ ]  trainX_eq=datasetHistogramEqualization(trainX_norm)
     valX_eq=datasetHistogramEqualization(valX_norm)
     testX_eq=datasetHistogramEqualization(testX_norm)
```

Στη συνέχεια θα κάνουμε data augmentations στο ήδη υπάρχον dataset με χρήση του image data generator του Keras.(Προσοχή θέλει στο γεγονός ότι η συνάρτηση flow του ImageDataGenerator δεν κάνει πρόσθεση εικόνων αλλά augmentation στις ήδη υπάρχουσες)

Τα augmentations που γίναν είναι τα εξής:

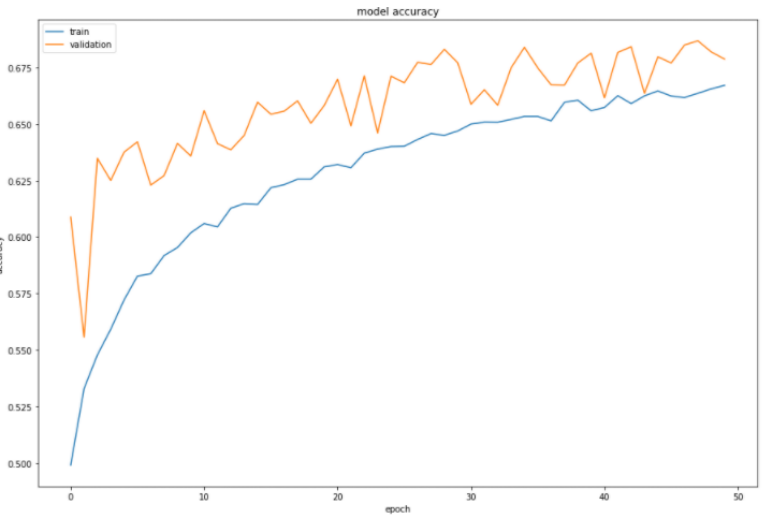
```
# Create an ImageDataGenerator and do Image Augmentation
training_datagen = ImageDataGenerator(
      rotation_range=30,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.1,
      horizontal_flip=True,
      fill_mode='nearest')
```

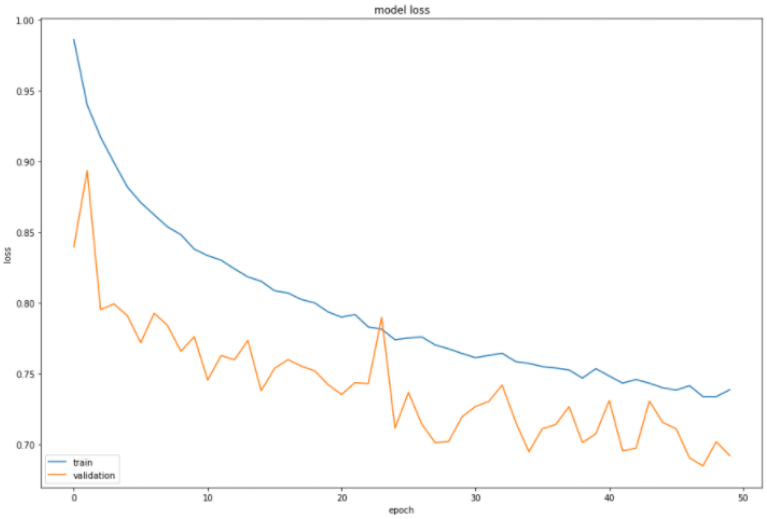
Οπότε με την προηγούμενη αρχιτεκτονική (διπλασιάζοντας και τα φίλτρα καθώς θα έχουμε σημαντική μείωση του overfit λόγω του data augmentation) αυτή είναι η τελική αρχιτεκτονική:

```
model = models.Sequential()

model.add(layers.Conv2D(16, (3, 3), input_shape=(64,64,1),padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Activation('swish'))

model.add(layers.Conv2D(32, (3, 3),padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Activation('swish'))

model.add(layers.Conv2D(32, (3, 3),padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Activation('swish'))
model.add(layers.MaxPooling2D(pool_size=(2,2)))

model.add(layers.Conv2D(64, (3, 3),padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Activation('swish'))

model.add(layers.Conv2D(64, (3, 3),padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Activation('swish'))
model.add(layers.MaxPooling2D(pool_size=(2,2)))

model.add(layers.Flatten())

# Fully connected layer
model.add(layers.Dense(256))
model.add(layers.Activation('swish'))
model.add(layers.Dropout(0.3))

model.add(layers.Dense(128))
model.add(layers.Activation('swish'))
model.add(layers.Dropout(0.3))

#add last layer for 3 classes
model.add(layers.Dense(3))
model.add(layers.Activation('softmax'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 16)        160
_________________________________________________________________
batch_normalization (BatchNo (None, 64, 64, 16)        64
_________________________________________________________________
activation (Activation)      (None, 64, 64, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 64, 32)        4640
_________________________________________________________________
batch_normalization_1 (Batch (None, 64, 64, 32)        128
_________________________________________________________________
activation_1 (Activation)    (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 32)        9248
_________________________________________________________________
batch_normalization_2 (Batch (None, 64, 64, 32)        128
_________________________________________________________________
activation_2 (Activation)    (None, 64, 64, 32)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
batch_normalization_3 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
activation_3 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 64)        36928
_________________________________________________________________
batch_normalization_4 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
activation_4 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 16384)             0
_________________________________________________________________
dense (Dense)                (None, 256)               4194560
_________________________________________________________________
activation_5 (Activation)    (None, 256)               0
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
activation_6 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 387
_________________________________________________________________
activation_7 (Activation)    (None, 3)                 0
=================================================================
Total params: 4,298,147
Trainable params: 4,297,731
Non-trainable params: 416
_________________________________________________________________
```

Κάνοντας την εκπαίδευση είχαμε τις καλύτερες καμπύλες μέχρι στιγμής





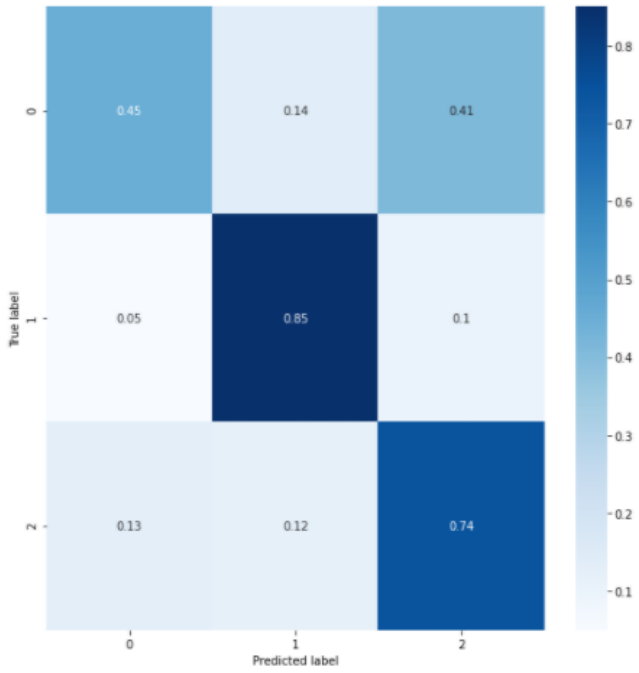
Παρατηρούμε ότι δεν έχουμε ούτε underfit ούτε overfit. (μάλιστα συνέχισα μετά την εκπαίδευση για άλλες 150 εποχές πριν δοκιμάσω στο test dataset)

Τα αποτελέσματα στο test dataset ήταν και τα καλύτερα από όλες τις δοκιμές:

Test Metrics
Classification Report for Test Dataset

	precision	recall	f1-score	support
0	0.69	0.45	0.54	1274
1	0.81	0.85	0.83	1721
2	0.60	0.74	0.66	1388
accuracy			0.70	4383
macro avg	0.70	0.68	0.68	4383
weighted avg	0.71	0.70	0.69	4383

Confusion Matrix



Τα αποτελέσματα είναι αρκετά καλά συγκριτικά με την δυσκολία του dataset.

3. Ερωτήσεις κατανόησης

a) Όπως είδαμε χαρακτηριστικά καθώς κάναμε τις υλοποιήσεις τα CNN είχαν μεγαλύτερη ακρίβεια. Ο βασικός λόγος που συμβαίνει αυτό είναι ότι τα CNN είναι καλύτερα όταν πρόκειται για ταξινόμηση εικόνων. Αυτό συμβαίνει καθώς στο ANN χάνουμε την spatial πληροφορία των pixel καθώς κάνουμε flatten το input και βλέπουμε το κάθε pixel ξεχωριστά. Αυτό σημαίνει ότι χάνουμε διαθέσιμη πληροφορία η οποία μάλιστα είναι πολύ σημαντική σημασιολογικά στην αναγνώριση του περιεχομένου μιας εικόνας. Στο CNN αντίθετα έχουμε την δυνατότητα να κάνουμε εσωτερικές αναπαραστάσεις δυσδιάστατων εικόνων βρίσκοντας έτσι «χωρικές» πληροφορίες. Επιπλέον εξαιτίας του τρόπου που δουλεύουν τα CNN μπορούμε να έχουμε μικρότερο αριθμό παραμέτρων για ίδιο βάθος δικτύου με το ANN επιτρέποντας μας να κάνουμε βαθύτερες υλοποιήσεις και να αναγνωρίζουμε υψηλού επιπέδου χαρακτηριστικά χωρίς να έχουμε overfit και πολύ αργό train.

b) Η βασική διαφορά είναι στο τρόπο που προσεγγίζουμε την εικόνα. Στο ANN το κάθε pixel αποτελεί ένα χαρακτηριστικό τελείως ξεχωριστό και εμείς προσπαθούμε με βάση τις εικόνες αυτές να διαχωρίσουμε σε κλάσεις αυτά τα χαρακτηριστικά.

Αντιθέτως το CNN αναγνωρίζει spatial relationships στο χώρο της εικόνας και μάλιστα ανεξαρτήτως της θέσης που βρίσκεται το χαρακτηριστικό (καθώς το filter περνάει όλη τη εικόνα) και εξαιτίας του pooling ανεξαρτήτως rotation/tilt. Κοινώς το μοντέλο στην εύρεση patterns είναι location invariant και local translation invariant. Τέλος όσο βαθύτερα πάμε στο δίκτυο μπορούμε συνδυάζοντας αυτά τα γεωμετρικά χαρακτηριστικά της εικόνας να ψάχνουμε για «υψηλού επιπέδου» patterns όπως περιγράψαμε παραπάνω. Όλα αυτά μαζί με αυτά που αναφέρθηκαν στο προηγούμενο ερώτημα αποτελούν την βασική διαφορά στην ταξινόμηση εικόνας με CNN αντί για ANN.

4. Bonus

Προετοιμασία Dataset

Το bonus dataset που μου δόθηκε ήταν το «Breast Histopathology Images» που περιγράψαμε στην εισαγωγή Αρχικά κάναμε την local προ επεξεργασία όπως και στο πρώτο dataset (με λίγο διαφορετικό τρόπο καθώς είναι αρχικώς αποθηκευμένα διαφορετικά). Οπότε με μερικές αλλαγές στην συνάρτηση που είχα κάνει για το αρχικό dataset φτιάχνουμε τα NumPy arrays του χωρισμένου dataset (με ίδια αναλογία 70/20/10 και αυτή τη φορά έγχρωμες και normalized εξαρχής με μέγεθος 50x50) .

Το dataset μας έχει τα παρακάτω χαρακτηριστικά:

```
train images shape (194001, 50, 50, 3)
validation images shape (55523, 50, 50, 3)
test images shape (28000, 50, 50, 3)
Train Images Per Class
Class 0 (Negative): 138985
Class 1 (Positive): 55016
Validation Images Per Class
Class 0 (Negative): 39757
Class 1 (Positive): 15766
Test Images Per Class
Class 0 (Negative): 19996
Class 1 (Positive): 8004
```

Παρατηρούμε ότι έχουμε ένα πολύ unbalanced dataset. Η τακτική για την εκπαίδευση σε αυτό το dataset είναι η ίδια με αυτήν που περιγράψαμε και πριν ,δηλαδή η χρήση class weights στο loss function.

```
class_weights = class_weight.compute_class_weight('balanced',
                                                  np.unique(trainY),
                                                  trainY)
class_weight_dict = dict(enumerate(class_weights))
class_weight_dict
```

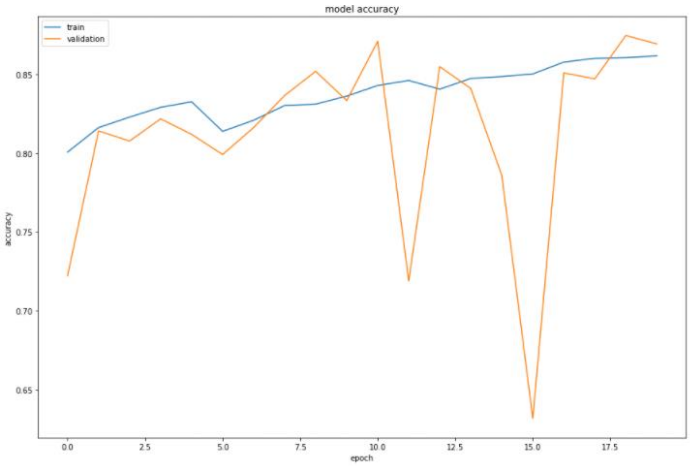
```
{0: 0.6979206389178688, 1: 1.7631325432601426}
```

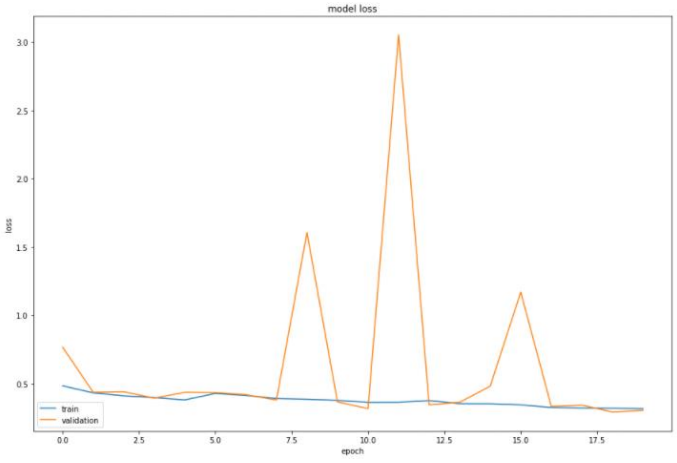
a)Θα φορτώσουμε το μοντέλο του resnet50 από το keras και θα το εκπαιδεύσουμε από την αρχή καθώς έχουμε πολύ μεγάλο dataset. Επιπλέον θα αλλάξουμε τα fully connected layers για να ταιριάζουν στην περίπτωση μας(θέλουμε binary crossentropy σε sigmoid για το τελευταίο layer καθώς έχουμε δυαδικό πρόβλημα) . Ο κώδικας για τα παραπάνω:

```
resnet50_model = applications.resnet50.ResNet50(weights= None, include_top=False, input_shape= (50,50,3))

model = models.Sequential()
model.add(resnet50_model)
model.add(layers.Flatten())
# Fully connected layer
model.add(layers.Dense(256))
model.add(layers.Activation('swish'))
model.add(layers.Dropout(0.3))

model.add(layers.Dense(128))
model.add(layers.Activation('swish'))
model.add(layers.Dropout(0.3))

#add last layer for 1 classes
model.add(layers.Dense(1))
model.add(layers.Activation('sigmoid'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Functional)        (None, 2, 2, 2048)        23587712
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 256)               2097408
_________________________________________________________________
activation (Activation)      (None, 256)               0
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
activation_1 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0
=================================================================
Total params: 25,718,145
Trainable params: 25,665,025
Non-trainable params: 53,120
```

Εκπαιδεύοντας για 20 εποχές είχαμε τα παρακάτω αποτελέσματα στο test dataset :

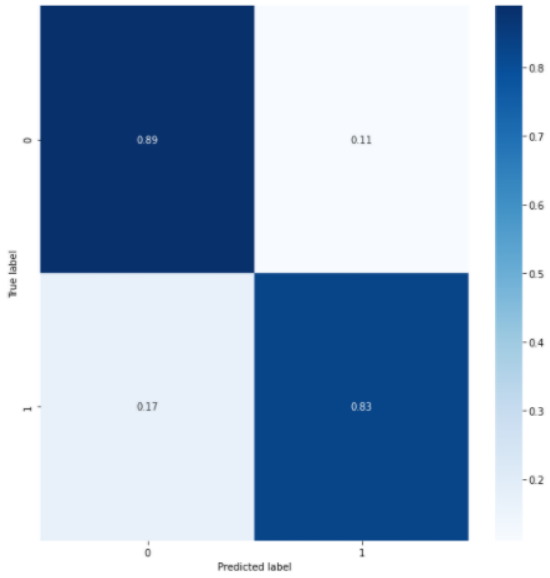




Test Metrics
Classification Report for Test Dataset

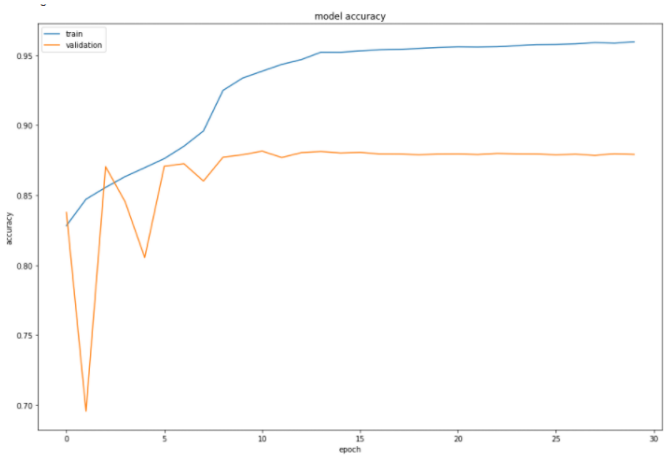
	precision	recall	f1-score	support
0	0.93	0.89	0.91	19996
1	0.75	0.83	0.79	8004
accuracy			0.87	28000
macro avg	0.84	0.86	0.85	28000
weighted avg	0.88	0.87	0.87	28000

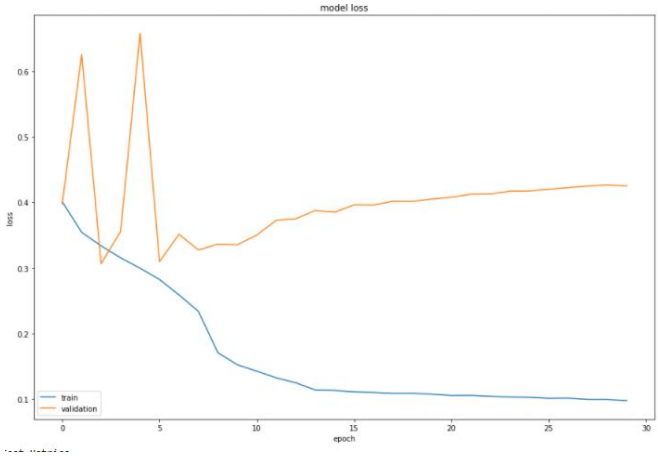
Confusion Matrix



Εδώ να πούμε ότι το accuracy δεν είναι καλή μετρική λόγω του πόσο unbalanced είναι το dataset.Ειδικά σε ιατρικό πρόβλημα είναι σημαντικό να βλέπουμε το recall και precision στα θετικά cases. Μπορούμε να δούμε πάντως ότι για απλό training είχαμε αρκετά καλά αποτελέσματα.

Τρέχοντας τώρα το ίδιο με το δικό μου μοντέλο για 20 εποχές είχαμε τα παρακάτω αποτελέσματα

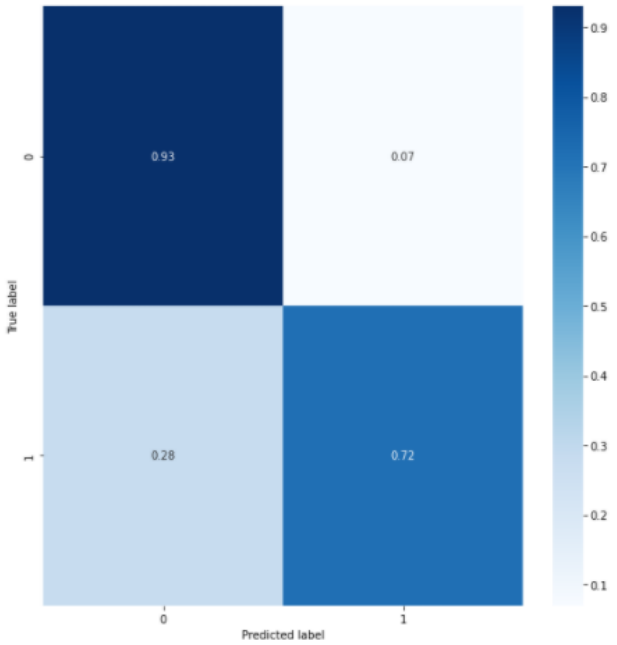




Test Metrics
Classification Report for Test Dataset

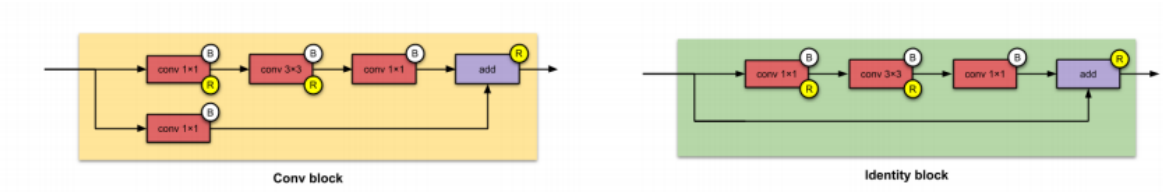
	precision	recall	f1-score	support
0	0.89	0.93	0.91	19996
1	0.80	0.72	0.76	8004
accuracy			0.87	28000
macro avg	0.85	0.83	0.84	28000
weighted avg	0.87	0.87	0.87	28000

Confusion Matrix



Τα αποτελέσματα είναι αρκετά κοντά ,το precision στα positive cases είναι καλύτερο αλλά το recall χειρότερο.

b) Η βασική διαφορά των ResNet σε σχέση με τα CNN είναι η ύπαρξη των residual block.



Επιπλέον εκτός αυτών των δομών δεν υπάρχουν και fully connected layers (στο original resnet)

c) το πρόβλημα που αντιμετωπίστηκε με το resnet είναι τα vanishing gradient που προκύπτει εξαιτίας του πολύ μεγάλου βάθους ενός δικτύου. Επιπλέον παρέχει άμεσα πληροφορία από πιο πίσω layers σε επόμενα. Δημιουργήθηκαν δηλαδή με σκοπό να μπορέσουμε να φτιάξουμε πιο βαθιά δίκτυα.

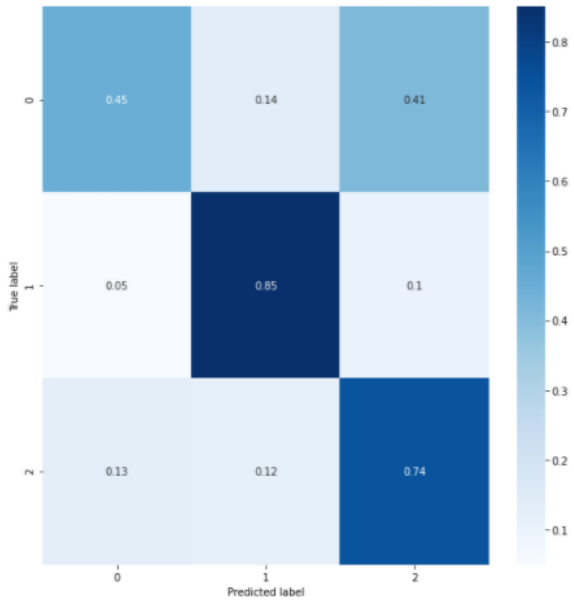
5. Ανακεφαλαίωση – Συζήτηση/Προτάσεις

Στην παρούσα εργασία όπως είδαμε λοιπόν αρχικά δοκιμάσαμε για το FER dataset τόσο τα CNN όσο και τα ANN μαζί με διάφορες τεχικές και τακτικές για την εκπαίδευση των δικτύων αυτών. Από τα αποτελέσματα βγάλαμε μερικά συμπεράσματα που δικαιολογήθηκαν με την αντίστοιχη θεωρία. Τα καλύτερα αποτελέσματα για το dataset αυτό ήταν όπως είδαμε τα παρακάτω:

Test Metrics
Classification Report for Test Dataset

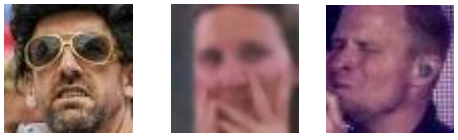
	precision	recall	f1-score	support
0	0.69	0.45	0.54	1274
1	0.81	0.85	0.83	1721
2	0.60	0.74	0.66	1388
accuracy			0.70	4383
macro avg	0.70	0.68	0.68	4383
weighted avg	0.71	0.70	0.69	4383

Confusion Matrix



Τα αποτελέσματα μπορεί να φαίνονται μέτρια αλλά το dataset ήταν αρκετά δύσκολο

Π.χ στο dataset:



Neutral

Disappointed

και θα ήταν ενδιαφέρον να δούμε ποιο είναι το αντίστοιχο human benchmark για αυτό το dataset.

Επιπλέον θα κάτι που δεν δοκιμάστηκε στην εργασία είναι κάποιο upscaling στο μέγεθος εικόνων πριν τις εισάγουμε στο νευρωνικά (π.χ. 128x128).

Επιπλέον ίσως κάποια ensemble τακτική για τον καλύτερο διαχωρισμό τον πιο μπερδεμένων κλάσεων (0:disappointed με 2: neutral) θα ήταν ενδιαφέρουσα.

Στο bonus dataset είδαμε τα ResNet και τα συγκρίναμε με ένα απλό CNN. Τα αποτελέσματα ήταν και για τις 2 αρχιτεκτονικές σχετικά καλά. Για να βελτιώσουμε το precision και recall για την underrepresented κλάση θα ήταν ενδιαφέρον να δοκιμάσουμε να κάνουμε άλλες μεθόδους αντί του class weight όπως resampling ή downsampling καθώς και δημιουργία συνθετικών παραδειγμάτων μέσω GAN's και να συγκρίνουμε τα αποτελέσματα