ΕΡΓΑΣΙΑ 4

ΟΡΑΣΗ ΥΠΟΛΟΓΙΣΤΩΝ

CONVOLUTIONAL NEURAL NETWORK

FOR TRAFFIC SIGNS DETECTION

ΟΝΟΜΑΤΕΠΩΝΥΜΟ:Ζαχάρη Βάια

ΑΜ:58161

ΕΙΣΑΓΩΓΗ:

Στη παρούσα εργασία κληθήκαμε να ασχοληθούμε με την επιστήμη των νευρωνικών δικτύων.Τα νευρωνικά δίκτυα είναι ένας κλάδος της επιστήμης,ο οποίος αν και έχει παρελθόν,βρίσκει εκθετική άνοδο τα τελευταία χρόνια.Εμείς ως φοιτητές πολυτεχνικής σχολής κληθήκαμε να ασχοληθούμε με τον κλάδο αυτόν,ο οποίος επιτρέπει ως ένα βαθμό τη δημιουργικότητα και την πρωτοβουλία μας. Αρχικά, παρουσιάζεται ένα νευρωνικό μοντέλο προσωπικής υλοποίησης και στη συνέχεια περιγράφεται ένα προεκπαιδευμένο μοντέλο. Σκοπός των δύο αυτών μοντέλων είναι να αναγνωρίζουν τις κατηγορίες(κλάσεις) των σημάτων οδικής κυκλοφορίας.Τέλος,τα μοντέλα θα εξεταστούν και σε εικόνες της προηγούμενης εργασίας,έτσι ώστε να συγκρίνουμε τις παλιές μεθόδους που εργασθήκαμε με τις πιο σύγχρονες.

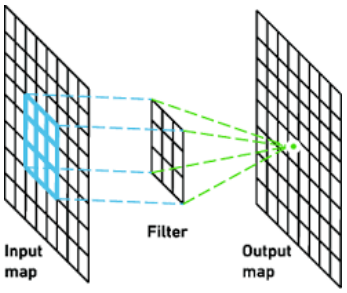
1ο ΜΕΡΟΣ

Πρωτού προχωρήσουμε στην αναλυτική περιγραφή της κάθε αρχιτεκτονικής, είναι ωφέλιμο να οριστούν έννοιες που θα χρησιμοποιηθούν.

- **Convolutional layer**:
  Ένα convolutional layer 2d λαμβάνει την εικόνα εισόδου και εκτελεί έναν κατά στοιχείο πολλαπλασιασμό των βαρών στον πυρήνα με τα αντίστοιχα στοιχεία στην εικόνα εισόδου. Η έξοδος του επιπέδου είναι ένας χάρτης χαρακτηριστικών, ο οποίος περιέχει τα εντοπισμένα χαρακτηριστικά. Στη συνέχεια, ο χάρτης χαρακτηριστικών χρησιμοποιείται στο επόμενο επίπεδο για την αναγνώριση αντικειμένων στην εικόνα.
  Οι βασικές παράμετροι που χρησιμοποιούνται είναι οι εξής:

```
tf.keras.layers.Conv2D(
  filters,
  kernel_size,
  strides=(1, 1),
  padding="same"
)
```

  - **Kernel size**: Δύο ακέραιοι, που καθορίζουν το ύψος και το πλάτος του παραθύρου συνέλιξης 2D.
  - **Strides**:Δύο ακέραιοι, που καθορίζουν τα βήματα της συνέλιξης κατά το ύψος και το πλάτος.
  - **Padding**: Το "same" έχει ως αποτέλεσμα την πλήρωση με μηδενικά ομοιόμορφα αριστερά/δεξιά ή πάνω/κάτω της εισόδου. Όταν padding="same" και strides=1, η έξοδος έχει το ίδιο μέγεθος με την είσοδο.



- **<u>BatchNormalization layer</u>**:Το επίπεδο BatchNormalization είναι ένας τύπος στρώματος που ομαλοποιεί τις ενεργοποιήσεις των νευρώνων του προηγούμενου στρώματος αφαιρώντας τον μέσο όρο της παρτίδας και διαιρώντας με την τυπική απόκλιση της παρτίδας. Αυτό προλαβαίνει την απότομη αύξηση των τιμών και άρα βοηθά στη βελτίωση της σταθερότητας του δικτύου. Διευκολύνει επίσης το μοντέλο να συγκλίνει ταχύτερα και βελτιώνει την ακρίβεια του μοντέλου.

  Οι βασικοί παράμετροι είναι :

  ```
  tf.keras.layers.BatchNormalization(
    axis=-1,
    epsilon=0.001)
  ```

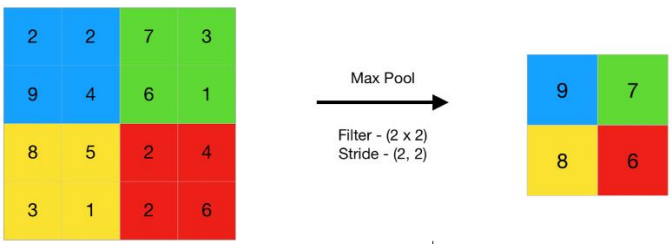
  - **Axis**: Ο άξονας που πρέπει να κανονικοποιηθεί (συνήθως ο άξονας των χαρακτηριστικών).
  - **Epsilon:** Μικρή διακύμανση για να αποφευχθεί η διαίρεση με το μηδέν.

- **<u>MaxPooling layer</u>:** Το max pooling είναι ένας τύπος στρώματος που είναι υπεύθυνο για τη μείωση του μεγέθους του χάρτη χαρακτηριστικών εισόδου λαμβάνοντας τη μέγιστη τιμή από την περιοχή του χάρτη.Κάνει δηλαδή υποδειγματοληψία. Αυτό βοηθά στη μείωση του όγκου των παραμέτρων στο μοντέλο και συμβάλλει στη μείωση του overfitting. Τοποθετείται μετά από convolutional layers για να μειωθεί το μέγεθος του χάρτη χαρακτηριστικών προτού μεταβιβαστεί στο επόμενο επίπεδο.

  Οι βασικές παράμετροι είναι:

  ```
  tf.keras.layers.MaxPooling2D(
  pool_size=(2, 2), strides=(2,2), padding="same")
  ```

  - **Pool_size:**Δύο ακέραιοι που δηλώνουν το μέγεθος του παραθύρου.,
  - **Strides:** Δύο ακέραιοι, που καθορίζουν πόσο μακρία κινείται το παράθυρο pooling σε κάθε βήμα.
  - **Padding:** Το "same" έχει ως αποτέλεσμα την πλήρωση με μηδενικά ομοιόμορφα αριστερά/δεξιά ή πάνω/κάτω της εισόδου. Όταν padding="same" και strides=1, η έξοδος έχει το ίδιο μέγεθος με την είσοδο.



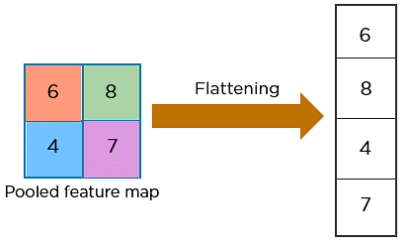
- **Activation layer(Relu):** Αυτό το στρώμα είναι υπεύθυνο για τον μετασχηματισμό του σήματος εισόδου εφαρμόζοντας τη συνάρτηση ανορθωμένης γραμμικής μονάδας. Η συνάρτηση ReLU επιστρέφει ο εάν η είσοδος είναι αρνητική και επιστρέφει την είσοδο εάν είναι θετική. Το επίπεδο ενεργοποίησης ReLU βοηθά στην επιτάχυνση της σύγκλισης του μοντέλου και επίσης βοηθά στην αποφυγή υπερβολικής προσαρμογής του μοντέλου.

- **Activation layer(Softmax):** Το στρώμα του είναι υπεύθυνο για τον μετασχηματισμό του σήματος εισόδου εφαρμόζοντας τη συνάρτηση softmax. Η συνάρτηση Softmax παράγει ένα διάνυσμα πιθανοτήτων που μπορεί να χρησιμοποιηθεί για την ταξινόμηση της εισόδου.

- **Dropout layer:** Αυτό το επίπεδο είναι υπεύθυνο για την τυχαία απόρριψη ορισμένου αριθμού νευρώνων του διανύσματος εισόδου. Το dropout layer βοηθά στην τακτοποίηση του μοντέλου και επίσης βοηθά στην αποφυγή overfitting του μοντέλου. Χρησιμοποιείται ως το τελευταίο στρώμα ενός νευρωνικού δικτύου, πριν από το επίπεδο εξόδου.

- **Dense layer:** Αυτό το επίπεδο είναι υπεύθυνο για τον υπολογισμό του σταθμισμένου αθροίσματος των εισόδων και την εφαρμογή μιας μη γραμμικής συνάρτησης ενεργοποίησης. Το dense layer βοηθά στην εκμάθηση πολύπλοκων μη γραμμικών σχέσεων μέσα στα δεδομένα. Τα βάρη του dense layer μπορούν να μάθουν κατά τη διάρκεια του training.

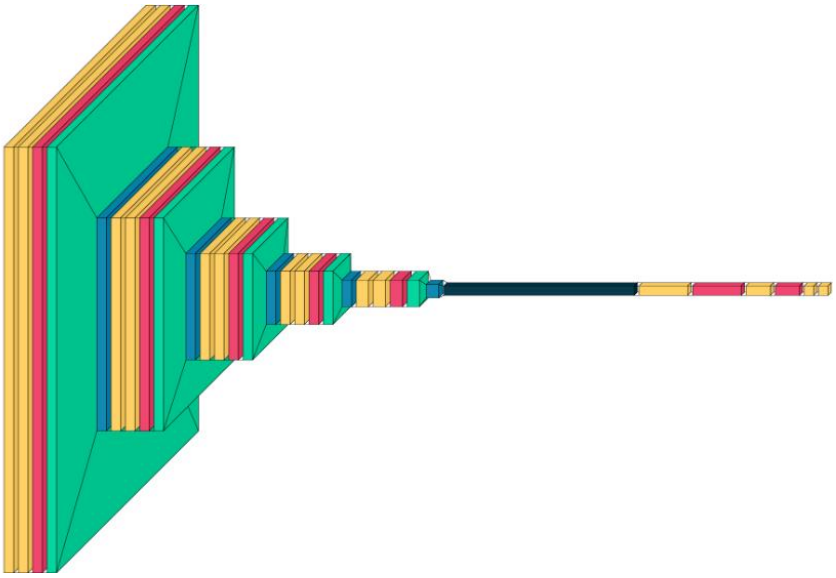
- **Flatten layer:** Τοποθετεί τα δεδομένα εισόδου σε ένα ενιαίο διάνυσμα. Αυτό επιτρέπει στο μοντέλο να χρησιμοποιεί τα χαρακτηριστικά που εξάγονται από τα cnnεπίπεδα fully connected επίπεδα.Είναι ένα σημαντικό μέρος πολλών μοντέλων deep learning καθώς επιτρέπει τη χρήση των ίδιων χαρακτηριστικών σε διαφορετικά επίπεδα του μοντέλου.



2ο ΜΕΡΟΣ:

Παρακάτω μπορούμε να παρατηρήσουμε την αρχιτεκτονική του CNN(Convolutional Neural Network) που σχεδίασθηκε.Για να αναπαραχθεί το σχήμα χρησιμοποιήθηκε η βιβλιοθήκη <visualkeras>.

Το αρχικό μέγεθος των εικόνων εισόδου είναι 224x224.



Όπως παρατηρούμε το CNN(Convolutional Neural Network) αποτελείται από:

- Δύο convolutional layers
- Ένα activation layer με τον αλγόριθμο Relu
- Ένα BatchNormalization layer
- Ένα Max Pooling layer

Αυτό το μοτίβο επαναλαμβάνεται πέντε φορές, όπου ο αριθμός και ο τύπος των kernel των αντίστοιχων convolution layers κάθε επιπέδου είναι τα εξης: (16, (5,5)), (32, (5,5)), (64,(5,5)), (128,(3,3)), (256,(3,3))

- Flatten layer
- Dense layer(1024)
- Dropout layer(0.2)
- Dense layer(512)
- Dropout layer(0.1)
- Dense layer(100)
- Dense layer(34)

Σημείωση: Το τελευταίο dense layer έχει 34 νευρώνες που καθορίζουν τις 34 διαθέσιμες κλάσεις του μοντέλου.

Διαχωρισμός συνόλου εικόνων

Στη συνέχεια για να διαχωρίσουμε τις εικόνες σε ομάδες δεδομένων εκπαίδευσης ,batches, χρησιμοποιούμε το ImageDataGenerator. Αυτό βοηθά στη μείωση των γενικών εξόδων επεξεργασίας ολόκληρου του συνόλου δεδομένων με μία κίνηση .Τελικά,επιλέχθηκε ως batch ο αριθμός 50.

Επιπλέον επειδή δεν διαθέτουμε ξεχωριστό φάκελο εικόνων για validation(=επιβεβαίωση), χρειάστηκε να σπάσουμε σε μέρη τον φάκελο με τις εικόνες εκπαίδευσης.Τα ποσοστά που επιλέχθηκαν είναι: 80% για το training και 20% για το validation. Αυτος ο διαχωρισμός εισάγεται ως παράμετρος στον ImageDataGenerator (validation_split=0.2). Για να είναι ακόμα πιο αξιόπιστο το σύνολο κάθε φορά ενεργοποιούμε τη μεταβλητή shuffle,βάση της οποίας οι εικόνες είναι ανακατεμένες.

Οπότε έχουμε δημιουργήσει δύο generators, train_generator και validation_generator,οι οποίοι παράγουν τα batches που χρειαζόμαστε για τη διαδικασία πριν το testing.

Επεξεργασία μοντέλου

Κατα τη διάρκεια της εκπαίδευσης του μοντέλου υπολογίζουμε ορισμένες μετρικές που μας υποδεικνύουν πόσο καλά το μοντέλο κινείται στη σωστή κατεύθυνση.

Συγκεκριμένα έχουμε το acc(η ακρίβεια),το val_acc(η διακύμανση της ακρίβειας),το loss(η απώλεια) και το val_loss (η διακύμανση της απώλειας). Επειδή αυτές οι μετρικές υπολογίζονται επαναληπτικά, με εποχές,για να μειώσουμε τις επαναλήψεις όταν αυτές δεν χρειάζονται, χρησιμοποιούμε δύο κλάσεις του keras.

- **EarlyStopping**: σταματάει τις εποχές(επαναλήψεις) όταν η μετρική val loss για 10(patience) εποχές δεν μειωθεί.

```
tf.keras.callbacks.EarlyStopping(
  monitor="val_loss",
  patience=10,
  verbose=1,
  restore_best_weights=True
```

)

- **ReduceLROnPlateau:** Μειώνει τον ρυθμό εκμάθησης σε 1e-6 από 1e-4, αν μετά από 4 εποχές δεν αλλάξει η μετρική val_loss.

```
tf.keras.callbacks.ReduceLROnPlateau(
  monitor="val_loss",
  factor=0.1,
  patience=4,
  verbose=1,
  min_lr=1e-6

)
```

Κάθε εποχή αποτελείται από ένα αριθμό βημάτων ο οποίος επηρεάζεται από τον αριθμό των batches. Όσο μεγαλύτερος είναι τόσο λιγότερα βήματα εκτελούνται.

```
steps_per_epoch=train_generator.samples/train_generator.batch_size
```

Επίσης επιλέγουμε μεγάλο αριθμό για τις εποχές,επειδή θέλουμε να βεβαιωθούμε ότι δεν υπήρχε καλύτερη επιλογή.
Παρακάτω φαίνοται αποτελέσματα ύστερα από 100 εποχές.

```
9s 186ms/step - loss: 5.9990e-05 - acc: 1.0000 - val_loss: 0.0535 - val_acc: 0.9900

9s 186ms/step - loss: 1.1632e-04 - acc: 1.0000 - val_loss: 0.0532 - val_acc: 0.9933

9s 188ms/step - loss: 4.9631e-05 - acc: 1.0000 - val_loss: 0.0537 - val_acc: 0.9917

9s 185ms/step - loss: 6.3659e-05 - acc: 1.0000 - val_loss: 0.0518 - val_acc: 0.9933
h as  jit compiled convolution op.  jit compiled convolution op.  jit compiled convolu
```

Testing

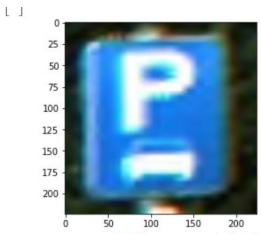
Κατα τη διαδικασία testing του μοντέλου χρησιμοποιούμε το φάκελο με τις εικόνες του testing ('imagedb_test').Με ρυθμό batch 50,κάνει αξιολόγη του μοντέλου ύστερα από την εκπαίδευση.
Τα ποσοστά που αναφέρονται παρακάτω σημαίνει ότι το μοντέλο έδειξε ακρίβεια 98.70% και είχε απώλεια μόλις 9.58%.

```
# --------------------
# Flow validation images in batches of 20 using test_datagen generator
# --------------------
test_generator =  test_datagen.flow_from_directory(test_dir,
                                                   batch_size=50,
                                                   class_mode='categorical',
                                                   #  color_mode='grayscale',
                                                    target_size=(224,224))
loss, acc = model.evaluate(test_generator)
```

```
Found 2149 images belonging to 34 classes.
43/43 [==============================] - 3s 64ms/step - loss: 0.0958 - acc: 0.9870
```

Δοκιμή μιας τυχαίας εικόνας

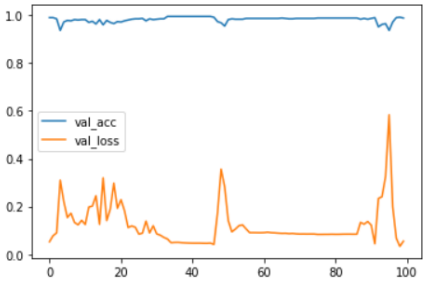
Στο τέλος, μπορούμε χειροκίνητα να επιλέξουμε τυχαία εικόνες από το φάκελο με τις εικόνες του test και να δούμε αν θα πετύχει τη σωστή κλάση.Ύστερα από διάφορες δοκιμές, παρατηρήθηκε ότι υπήρχαν παραπάνω ευστοχίες παρά αστοχίες.



```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<
1/1 [==============================] - 0s 158ms/step

['00004', '00005', '00007', '00008', '00010', '00012', '00013', '00017', '00018', '0001

Softmax Output: [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]]

/content/imagedb_test/00047/00069_00000.ppm is a 00047
```

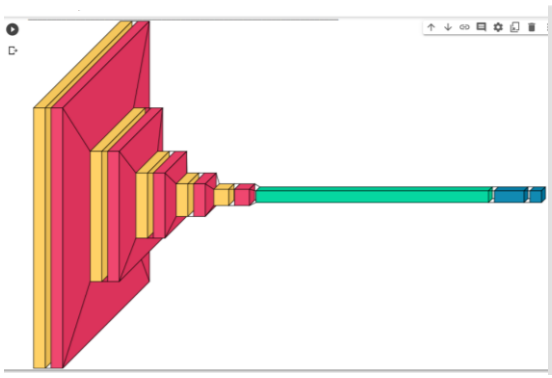
```
plt.plot(history.history['val_acc'],label='val_acc')
plt.plot(history.history['val_loss'],label='val_loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fc348fd1e80>
```



[Μια πιο απλή εκδοχή της παραπάνω αρχιτεκτονικής]

Προφανώς η παραπάνω αρχιτεκτονική δεν προέκυψε με τη πρώτη δοκιμή.Χρειάστηκαν αρκετές δοκιμές παραμέτρων, επιπέδων για να καταλήξουμε στη τελική.Κύριο κριτήριο αποτελούσε η διακύμανση της ακρίβειας του training και η ακρίβεια του testing.Ένα πιο απλό παράδειγμα είναι το εξής:



```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import activations
import visualkeras

model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(16, (5, 5), strides=(2, 2), input_shape=(224, 224, 3),padding='same'),
  tf.keras.layers.Activation(activations.relu),
  tf.keras.layers.Conv2D(32, (5, 5), strides=(2, 2),padding='same'),
  tf.keras.layers.Activation(activations.relu),
  tf.keras.layers.Conv2D(64, (5, 5), strides=(2, 2),padding='same'),
  tf.keras.layers.Activation(activations.relu),
  tf.keras.layers.Conv2D(128, (3, 3), strides=(2, 2),padding='same'),
  tf.keras.layers.Activation(activations.relu),
  tf.keras.layers.Conv2D(256, (3, 3), strides=(2, 2),padding='same'),
  tf.keras.layers.Activation(activations.relu),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dense(34, activation='softmax')

])
# Show a summary of the model. Check the number of trainable parameters
model.summary()
visualkeras.layered_view(model)
```

Τα αποτελέσματα του testing ήταν χειρότερα από τα παραπάνω.

```
test_datagen  = ImageDataGenerator(rescale=1./255)

# --------------------
# Flow validation images in batches of 20 using test_datagen generator
# --------------------
test_generator =  test_datagen.flow_from_directory(test_dir,
                                                   batch_size=50,
                                                   class_mode='categorical',
                                                   #  color_mode='grayscale',
                                                    target_size=(224,224))
loss, acc = model.evaluate(test_generator)
```

```
Found 2149 images belonging to 34 classes.
43/43 [==============================] - 2s 50ms/step - loss: 0.6282 - acc: 0.9288
```

Ακρίβεια:92.88%

Ο λόγος είναι ότι τα επιπλεόν επίπεδα που προσθέτουμε,όπως το batchnornalization, maxpooling, dropout βοηθούν στη βελτιστοποίηση των πόρων και στον πιο επιτυχημένο τρόπο να επιλέγει τα βάρη και τους ενεργούς νευρώνες κάθε φορά. Σε περίπτωση που είχαμε ένα πιο περίπλοκο νευρωνικό δίκτυο τότε η διαφορά στα αποτελέσματα θα ήταν πιο μεγάλη.

Augmentation

Το augmentation,δηλαδή η εφαρμογή μετασχηματισμών στις εικόνες είναι μία μέθοδος που υλοποιείται με τη βοήθεια του ImagaDataGenerator. Ήδη έχει αναφερθεί μία βασική παράμετρος,όπως το split. Τώρα βέβαια θα αναφερθούμε σε μετασχηματισμούς που αφορούν περισσότερο την γενική επεξεργασία της κάθε εικόνας.

Αρχικά μπορούμε να χρησιμοποιήσουμε:

-zoom_range = 0.1 Μεγένθυση κατα 10%

-horizontal_flip=True Περιστροφή οριζόντια

-rotation_range =(-15,+15) Περιστροφή δεξιά και αριστερα για λίγες μοίρες.

Γενικά είναι δύσκολο να καταλάβουμε πλήρως τις αναγκές του μοντέλου για augmentation,καθώς αυτοί οι μετασχηματισμοί εφαρμόζονται σε όλες τις εικόνες,ενώ όλες οι εικόνες δεν έχουν τα ίδια χαρακτηριστικά. Έτσι μία εικόνα μπορεί να έχει αρκετή φωτεινότητα ,επειδή το σήμα είναι εκτεθειμένο στον ήλιο, είτε ένα σημα να έχει φωτογραφηθεί βράδυ.

```
==========] - 30s 618ms/step - loss: 0.0051 - acc: 0.9988 - val_loss: 0.0717 - val_acc: 0.9833
==========] - 31s 620ms/step - loss: 0.0060 - acc: 0.9976 - val_loss: 0.1193 - val_acc: 0.9766
==========] - 31s 623ms/step - loss: 0.0029 - acc: 0.9996 - val_loss: 0.0923 - val_acc: 0.9850
==========] - 30s 620ms/step - loss: 5.5372e-04 - acc: 1.0000 - val_loss: 0.0941 - val_acc: 0.9783
==========] - 30s 617ms/step - loss: 0.0011 - acc: 1.0000 - val_loss: 0.0770 - val_acc: 0.9866
==========] - 30s 619ms/step - loss: 0.0023 - acc: 0.9992 - val_loss: 0.0668 - val_acc: 0.9883
ıntraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolutic
```

```
test_datagen  = ImageDataGenerator(rescale=1./255)

# --------------------
# Flow validation images in batches of 20 using test_datagen generator
# --------------------
test_generator =  test_datagen.flow_from_directory(test_dir,
                                                   batch_size=50,
                                                   class_mode='categorical',
                                                   #  color_mode='grayscale',
                                                    target_size=(224,224))
loss, acc = model.evaluate(test_generator)
```

```
Found 2149 images belonging to 34 classes.
43/43 [==============================] - 3s 63ms/step - loss: 0.1092 - acc: 0.9879
```

Ακρίβεια:98.79%

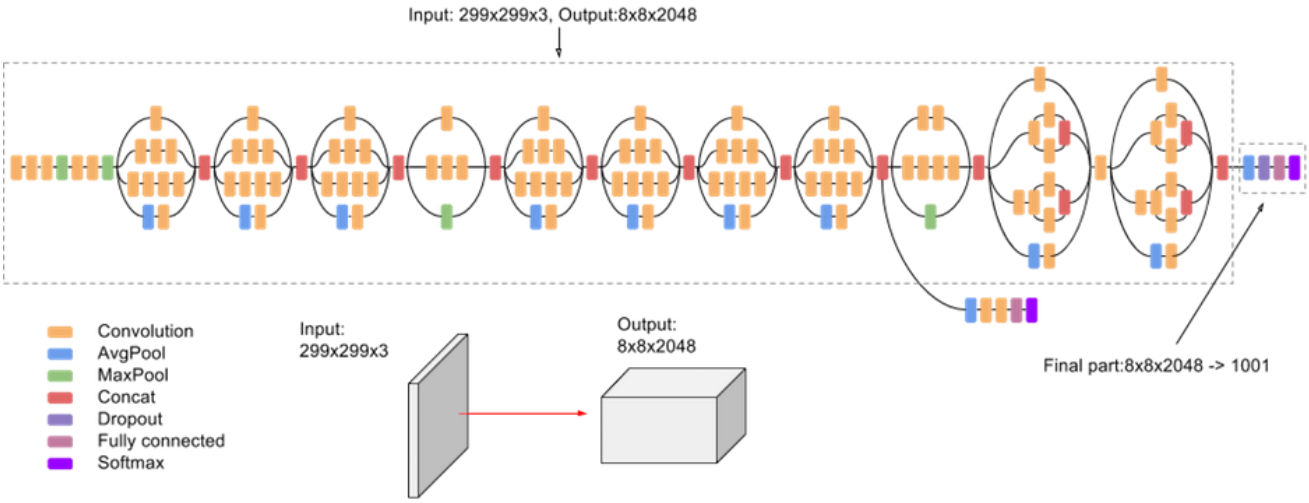
3ο Μέρος

Inceptionv3

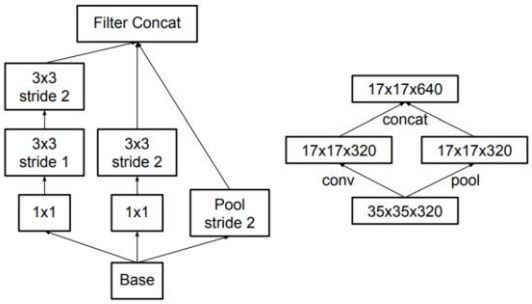
Στη συγκεκριμένη εργασία επιλέχθηκε ως προεκπαιδευμένο νευρωνικό το InceptionV3,που αναπτύχθηκε από την Google το 2015 .Το όνομα του CNN Inception έχει ιδιαίτερο ενδιαφέρον,καθώς παραπέμπει στην ομώνυμη ταινία. Η αναφορά που τα συνδέει είναι ότι και στις δύο περιπτώσεις η καλύτερη απόδοση προκύπτει με τις πιο πολλές μεταβάσεις από επίπεδο σε επίπεδο,τη πιο βαθιά μάθηση.

Οι λόγοι που επιλέχθηκε το συγκεκριμένο νευρωνικό είναι η ταχύτητα και το μικρό του μέγεθος σε συνδυασμό με την απόδοση του. Ωστόσο έχει πιο σύνθετη αρχιτεκτονική.Χρησιμοποιείται σε αρκετές εφαρμογές,όπως αναγνώριση αντικειμένων,προσώπων, μοτίβων.

Αρχιτεκτονικη InceptionV3:



- **Factorized Convolutions**: Αυτό βοηθά στη μείωση της υπολογιστικής απόδοσης καθώς μειώνει τον αριθμό των παραμέτρων που εμπλέκονται σε ένα δίκτυο. Επίσης, ελέγχει την αποτελεσματικότητα του δικτύου.
- **Μικρότερα Convolutional**: η αντικατάσταση μεγαλύτερων convolutional με μικρότερα οδηγεί σίγουρα σε ταχύτερη εκπαίδευση. Αν διαθέτουμε ένα φίλτρο 5 × 5 έχει 25 παραμέτρους. Δύο φίλτρα 3 × 3 που αντικαθιστούν μια συνέλιξη 5 × 5 έχουν μόνο 18 (3*3 + 3*3) παραμέτρους.
- **Ασύμμετρα Convolutional**:Ένα 3 × 3 θα μπορούσε να αντικατασταθεί από ένα 1 × 3 ακολουθούμενο από ένα 3 × 1. Εάν μια συνέλιξη 3 × 3 αντικατασταθεί από μια συνέλιξη 2 × 2, ο αριθμός των παραμέτρων θα είναι ελαφρώς υψηλότερος από την προτεινόμενη ασύμμετρη συνέλιξη.
- **Auxiliary classifier**: ένας βοηθητικός ταξινομητής είναι ένα μικρό CNN που εισάγεται μεταξύ των επιπέδων κατά τη διάρκεια της εκπαίδευσης και η απώλεια που προκύπτει προστίθεται στην απώλεια του κύριου δικτύου. Στο GoogLeNet, χρησιμοποιήθηκαν για ένα βαθύτερο δίκτυο, ενώ στο Inception v3 ένας βοηθητικός ταξινομητής λειτουργεί για κανονικοποίηση.(softmax layer)
- **Grid size reduction-Concat**(Μείωση μεγέθους πλέγματος): Η μείωση του μεγέθους του πλέγματος γίνεται συνήθως με λειτουργίες ομαδοποίησης. Ωστόσο, για την καταπολέμηση των σημείων συμφόρησης του υπολογιστικού κόστους, προτείνεται μια πιο αποτελεσματική τεχνική:



Στο τελευταίο βήμα,παρόλο που λαμβάνουμε τα βάρη στην αρχή, προστέθηκαν επιπλέον layers,ώστε να βελτιστοποιήσουμε την απόδοση και να ορίσουμε το πλήθος των κλάσεων του προβλήματος.

```
# Create the model
model = models.Sequential()

# Add the inception convolutional base model
model.add(modelVaia)

# Add new layers
model.add(layers.Flatten())
model.add(layers.Dense(2048, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(1024, activation='relu'))
if dataset_4:
 model.add(layers.Dense(34, activation='softmax'))
else:
  model.add(layers.Dense(5, activation='softmax'))

# Show a summary of the model. Check the number of trainable parameters
model.summary()
```

- **Χωρις augmentation,pretrained:**

```
Epoch 26/80
49/49 [==============================] - 30s 619ms/step - loss: 0.2052 - acc: 0.9349 - val_loss: 0.2546 - val_acc: 0.9282
Epoch 27/80
49/49 [==============================] - 30s 629ms/step - loss: 0.1762 - acc: 0.9369 - val_loss: 0.2335 - val_acc: 0.9349
Epoch 28/80
```

```
test_datagen  = ImageDataGenerator(preprocessing_function=preprocess_input)

# --------------------
# Flow validation images in batches of 20 using test_datagen generator
# --------------------
test_generator =  test_datagen.flow_from_directory(test_dir,
                                                   batch_size=50,
                                                   class_mode='categorical',
                                                   #  color_mode='grayscale',
                                                    target_size=(224,224))
loss, acc = model.evaluate(test_generator)
```

```
Found 2149 images belonging to 34 classes.
43/43 [==============================] - 5s 112ms/step - loss: 0.2971 - acc: 0.9390
```

Ακρίβεια:93.90%

- **Με augmentation,pretrained:**

```
Epoch 76/80
49/49 [==============================] - 33s 679ms/step - loss: 0.0539 - acc: 0.9829 - val_loss: 0.1827 - val_acc: 0.9533
Epoch 77/80
49/49 [==============================] - 32s 652ms/step - loss: 0.0685 - acc: 0.9776 - val_loss: 0.2102 - val_acc: 0.9466
Epoch 78/80
49/49 [==============================] - 34s 696ms/step - loss: 0.0701 - acc: 0.9772 - val_loss: 0.2460 - val_acc: 0.9449
Epoch 79/80
49/49 [==============================] - 32s 656ms/step - loss: 0.1061 - acc: 0.9703 - val_loss: 0.2414 - val_acc: 0.9416
Epoch 80/80
49/49 [==============================] - 32s 652ms/step - loss: 0.0867 - acc: 0.9735 - val_loss: 0.2113 - val_acc: 0.9432
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _j
```

```
test_datagen  = ImageDataGenerator(preprocessing_function=preprocess_input)

# --------------------
# Flow validation images in batches of 20 using test_datagen generator
# --------------------
test_generator =  test_datagen.flow_from_directory(test_dir,
                                                   batch_size=50,
                                                   class_mode='categorical',
                                                   #  color_mode='grayscale',
                                                    target_size=(224,224))
loss, acc = model.evaluate(test_generator)
```

```
Found 2149 images belonging to 34 classes.
43/43 [==============================] - 5s 119ms/step - loss: 0.2818 - acc: 0.9353
```

Ακρίβεια:93.53%

4ο Μέρος

Εφαρμογή των μοντέλων στα dataset της εργασίας3

- **Χωρίς augmentation,non-pretrainable cnn**

```
Epoch 97/100
7/7 [==============================] - 2s 219ms/step - loss: 8.4858e-05 - acc: 1.0000 - val_loss: 0.1829 - val_acc: 0.9574
Epoch 98/100
7/7 [==============================] - 2s 220ms/step - loss: 6.5971e-05 - acc: 1.0000 - val_loss: 0.1792 - val_acc: 0.9574
Epoch 99/100
7/7 [==============================] - 2s 219ms/step - loss: 4.2821e-05 - acc: 1.0000 - val_loss: 0.1770 - val_acc: 0.9574
Epoch 100/100
7/7 [==============================] - 2s 217ms/step - loss: 4.0436e-05 - acc: 1.0000 - val_loss: 0.1764 - val_acc: 0.9574
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op,
```

```
#test_datagen  = ImageDataGenerator(rescale=1./255)
test_datagen  = ImageDataGenerator()
# --------------------
# Flow validation images in batches of 20 using test_datagen generator
# --------------------
test_generator =  test_datagen.flow_from_directory(test_dir,
                                                   batch_size=50,
                                                   class_mode='categorical',
                                                   #  color_mode='grayscale',
                                                    target_size=(224,224))
loss, acc = model.evaluate(test_generator)
```

Με πολύ καλή ευστοχία.

- **Με augmentation,non-pretrainable cnn**

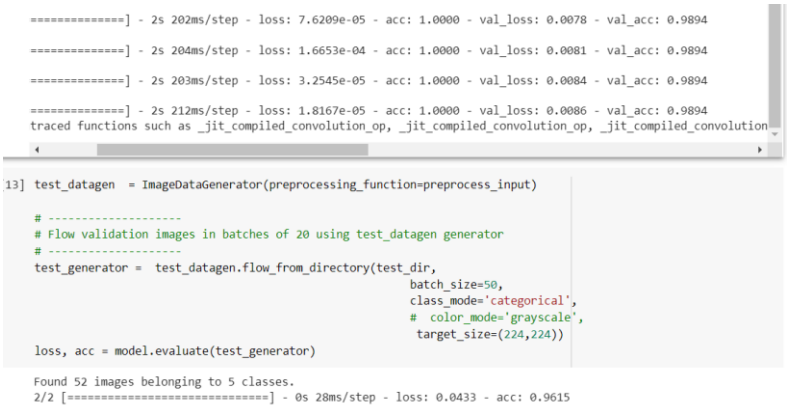
```
Epoch 96/100
7/7 [==============================] - 6s 730ms/step - loss: 0.0541 - acc: 0.9820 - val_loss: 0.3159 - val_acc: 0.8723
Epoch 97/100
7/7 [==============================] - 6s 726ms/step - loss: 0.0367 - acc: 0.9845 - val_loss: 0.8412 - val_acc: 0.7979
Epoch 98/100
7/7 [==============================] - 6s 740ms/step - loss: 0.0526 - acc: 0.9845 - val_loss: 0.2890 - val_acc: 0.9468
Epoch 99/100
7/7 [==============================] - 6s 729ms/step - loss: 0.0099 - acc: 0.9974 - val_loss: 0.2574 - val_acc: 0.9362
Epoch 100/100
7/7 [==============================] - 6s 754ms/step - loss: 0.0126 - acc: 0.9974 - val_loss: 0.1658 - val_acc: 0.9468
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_c
```

```
[6] #test_datagen  = ImageDataGenerator(rescale=1./255)
    test_datagen  = ImageDataGenerator()
    # --------------------
    # Flow validation images in batches of 20 using test_datagen generator
    # --------------------
    test_generator =  test_datagen.flow_from_directory(test_dir,
                                                       batch_size=50,
                                                       class_mode='categorical',
                                                       #  color_mode='grayscale',
                                                        target_size=(224,224))
    loss, acc = model.evaluate(test_generator)
```

```
Found 52 images belonging to 5 classes.
2/2 [==============================] - 0s 290ms/step - loss: 0.3272 - acc: 0.9423
```

Με πολύ καλή ευστοχία.Ακρίβεια: 94.23%

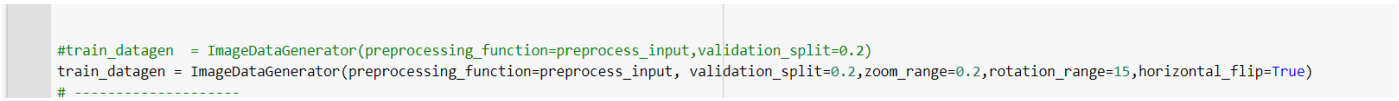
- **Χωρις augmentation pretrained model**

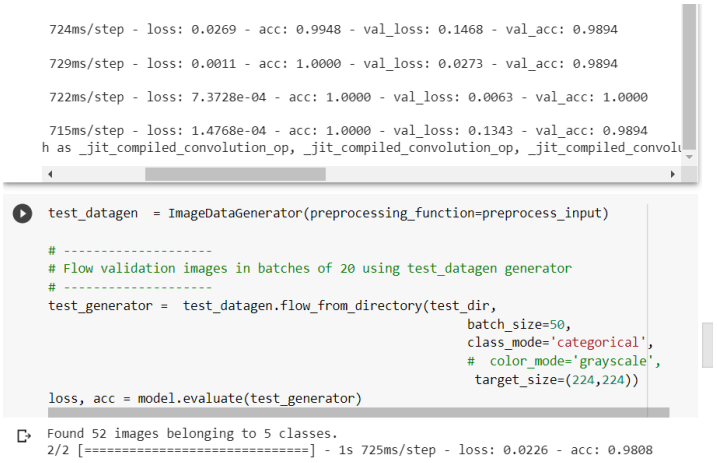


Ακρίβεια:96.16%

Με πολύ καλή ευστοχία,με εξαίρεση τη κλάση των μοτοσυκλετών.Αυτό το φαινόμενο είχε παρατηρηθεί και στη προηγούμενη εργασία.Ωστόσο το ποσοστό επιτυχίας εδώ είναι σαφώς πιο μεγάλο.

- **Με augmentation,pretrained model:**





Ακρίβεια:98.08%

Παρατηρώ ότι με augmentation αυξήθηκε η ακριβεία.Ωστόσο παρατηρήθηκαν αστοχίες στις χειροκινήτες δοκιμές.Συγκεκριμένα εμφάνιζε τη κλάση school bus στις περισσότερες κλάσεις

Συμπεράσματα-Παρατηρήσεις:

Το προεκπαιδευμένο μοντέλο που χρησιμηποιήθηκε παρουσιάζει αστοχίες στην εύρεση των τυχαίων εικόνων(στο τελευταίο block) κάτι που οφείλεται στην προεπεξεργασία που χρειάζεται η εικόνα πριν φορτωθεί.Αρχικά χρειάζεται resize σε (299*299) pixel και επιπλέον χρειάζεται να αφαιρεθεί ο μέσος όρος από κάθε κανάλι χρώματος, καθώς αυτό βοηθά στην ελαχιστοποίηση τυχόν παραμορφώσεων που προκαλούνται από διαφορές στις συνθήκες φωτισμού. Επιπλέον, οι εικόνες θα πρέπει επίσης να κανονικοποιούνται μεταξύ 0 και 1.Δυστυχώς δεν μπόρεσα να το προσθέσω αυτό το preprocessing κομμάτι στη τελευταία διαδικασία και για αυτό έχω τις τυχόν αστοχίες.Στη διαδικασία training και testing το preprocessing γίνεται μέσω του ImageDataGenerator εισάγοντας την preprocessing function.

Επιπλέον ύστερα από δοκιμές μπορούμε να καταλήξουμε ότι αυξάνοντας τις εποχές,αυξάναμε τη πιθανότητα να πετύχουμε καλύτερη ακρίβεια,καθώς με τις πολλές επαναλήψεις τα βάρη ανανεώνονται ,αλλά έχοντας ως βάση τα παλιά βάρη.Έτσι έχουμε διαρκώς βελτίωση.Σε περίπτωση που αυτή η βελτίωση πάψει να υπάρχει, σταματά και η διαδικασία του training. Επιπρόσθετα, αυξάνοντας τον αριθμό των batches ,όπως προαναφέρθηκε, αυξάνονται τα βήματα σε μία επανάληψη και άρα μεταβαίνουμε πιο ομαλά στις τιμές που θα μας επιφέρουν καλύτερα αποτελέσματα.Εκεί,δηλαδή, όπου η μετρική val_loss δεν μειώνεται άλλο.

Καταληκτικά,θα ήθελα να επισημάνω το ενδιαφέρον μου σχετικά με την όλη εργασία,αρχικά γιατί άπτεται σε σύγχρονες εφαρμογές και κατα δεύτερον διότι δίνει αρκετά ερεθίσματα για παραπάνω ενασχόληση με το αντικείμενο.