Όραση Υπολογιστών

Εργασία 4

Ακαδημαϊκό Έτος: 2019-2020

Καρασακαλίδης Αλέξανδρος Ηλίας

ΑΜ: 57448

Ζητούμενα

Η άσκηση αφορά στην υλοποίηση ενός συνελικτικού νευρωνικού δικτύου για την ταξινόμηση εικόνων.

Για την εκπόνηση της εργασίας θα χρησιμοποιηθεί η πλατφόρμα Kaggle, όπου έχει δημιουργηθεί ο εξής InClass διαγωνισμός:

https://www.kaggle.com/t/dd4081b7eabd4687be097654099114fb

όπου θα υποβληθούν οι υλοποιήσεις μας, υπο την μορφή notebook.

Βασική προυπόθεση για την υποβολή της υλοποίησής μας είναι να υποβληθεί μη προ-εκπαιδευμένο δίκτυο.

Μας δίνονται τρία υποσύνολα (training, validation και testing), στα οποία περιέχονται εικόνες με τα εξής αντικείμενα:

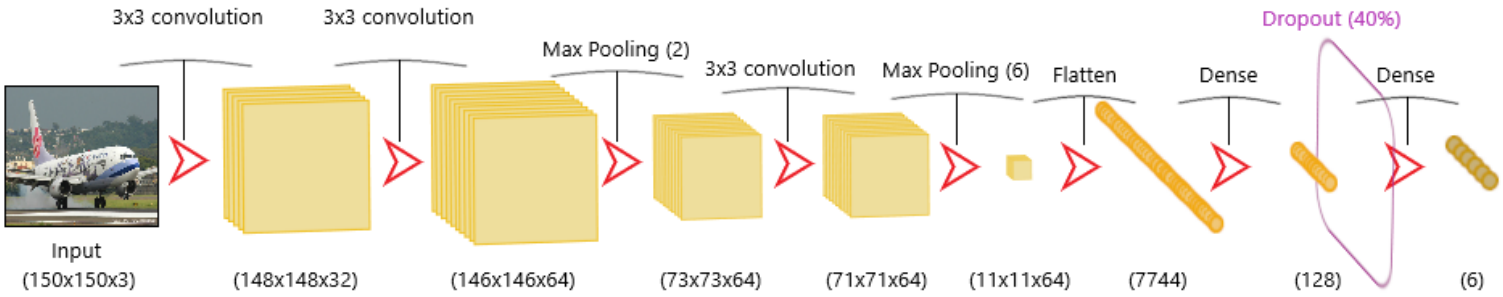
Αεροπλάνα	[1η Κλάση]
Ποδήλατα	[2η Κλάση]
Αυτοκίνητα	[3η Κλάση]
Ελικόπτερα	[4η Κλάση]
Μοτοσυκλέτες	[5η Κλάση]
Σχολικά Λεωφορεία	[6η Κλάση]

Τα αποτελέσματα θα πρέπει να παραδοθούν υπό μορφή csv, όπως φαίνεται στην διπλανή εικόνα.

009_0097.jpg	2
009_0098.jpg	2
158_0145.jpg	0
009_0095.jpg	0
158_0133.jpg	2
009_0102.jpg	0
009_0100.jpg	1
158_0140.jpg	2
009_0101.jpg	2
158_0138.jpg	2
009_0092.jpg	1
158_0132.jpg	0

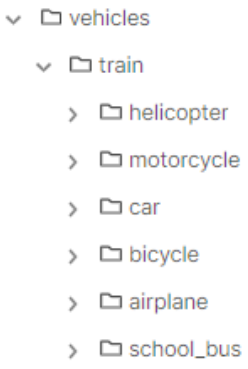
Υλοποίηση

Κατά την διάρκεια της υλοποίησης του δικτύου, δοκιμάστηκαν πολλοί διαφορετικοί συνδυασμοί Convolution, Max Pooling και Dense layers εκ των οποίων 4 συνδυασμοί έδωσαν ικανοποιητικά αποτελέσματα. Τελικά, το δίκτυο που επιλέχθηκε, έχει την εξής δομή:



Μεταξύ άλλων, βαθύτερων δικτύων που επιλέχθηκαν (με ίδιες παραμέτρους πέρα του συνόλου των layers), το παραπάνω δίκτυο απέδωσε τα καλύτερα αποτελέσματα.

Ξεκινάμε με τα δεδομένα μας. Όπως μας έχει υποδειχθεί έχουμε να κατηγοριοποιήσουμε ένα σύνολο εικόνων ανάμεσα σε έξι κλάσεις. Το training και το validation set που μας δόθηκαν είναι ήδη ταξινομημένα μέσα σε αντίστοιχους φάκελους όπως φαίνεται στο σχήμα.



Το training set αποτελείται από 2.494 εικόνες ενώ το validation από 311 εικόνες.

Η φόρτωση των δεδομένων και η προ-επεξεργασία τους έγινε με generators. Οι generators αυτοί ενώ έδιναν σε ομάδες (batches) τις εικόνες από τους φακέλους, εφάρμοζαν ταυτόχρονα μικρούς, τυχαίας κλίμακας μετασχηματισμούς στις εικόνες αυτές.

```
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale = 1. / 255, rotation_range = 30, zoom_range = 0.25, width_shift_range = 0.2,
                                                                height_shift_range = 0.2, horizontal_flip = True)

val_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale = 1. / 255)

validation_generator = val_datagen.flow_from_directory(
  val_dir,
  target_size = (150, 150),
  batch_size = 100,
  class_mode = 'categorical')

train_generator = train_datagen.flow_from_directory(
  train_dir,
  target_size = (150, 150),
  batch_size = bs,
  class_mode = 'categorical',
  shuffle = True)
```

Συγκεκριμένα, ο generator για το training set εφάρμοζε σε τυχαία κλίμακα τους εξής μετασχηματισμούς:

- Περιστροφή έως 30 μοίρες
- Εστίαση έως 25%
- Οριζόντια μετατόπιση κατά 20% της εικόνας
- Κατακόρυφη μετατόπιση κατά 20% της εικόνας
- Οριζόντιος καθρεπτισμός της εικόνας

Επίσης για κάθε εικόνα (validation set και μη), κανονικοποιούνται οι τιμές των εικονοστοιχείων της στα τρία κανάλια (RGB) που μας δίνονται. Τέλος, δίνονται οι εικόνες στους generators με την συνάρτηση flow_from_directory, απ'όπου και ξεχωρίζονται οι κλάσεις εικόνων με βάση τον φάκελο που προέρχονται. Σημαντικές παράμετροι των generator είναι το target_size και το batch_size.

Το target_size ορίζει τις διαστάσεις στις οποίες οι εικόνες θα μετασχηματισούν, ώστε να δοθούν σαν είσοδος στο δίκτυο. Όσο μεγαλύτερες οι διαστάσεις της εικόνας, τόσο βαθύτερο θα πρέπει να είναι το δίκτυο ώστε να μπορεί να ανταποκριθεί αποτελεσματικά στην αύξηση του όγκου πληροφορίας στην είσοδο. Ένας καλός συμβιβασμός φάνηκε ότι είναι οι διαστάσεις 150x150.

Το batch_size ορίζει τις ομάδες των εικόνων με τις οποίες προπονείται σε κάθε βήμα το δίκτυο. Τον αριθμό των εικόνων δηλαδή πάνω στον οποίο προπονείται κάθε φορά, πριν ενημερώσει τα βάρη του. Η επιλογή τιμής για το batch_size δικαιολογείται παρακάτω.

Φάνηκε ότι τα δεδομένα με τα οποία θα προπονούταν το δίκτυο δεν ήταν ισορροπημένα. Για παράδειγμα, οι εικόνες που απεικονίζουν ελικόπτερα ήταν έως και 3 φορές λιγότερες σε σχέση με τις υπόλοιπες κλάσεις. Μια λύση σε αυτόν τον περιορισμό, ήταν το να εισάγουμε βάρη (κατά την προπόνηση) στις κλάσεις, ώστε να μειωθεί το αντίκτυπο αυτής της ανισορροπίας. Θα αναλυθεί περισσότερο όμως παρακάτω. Προς το παρόν, τα βάρη με βάση την ανισορροπία, υπολογίζονται από την συνάρτηση:

```
class_weights = class_weight.compute_class_weight(
                'balanced',
               np.unique(train_generator.classes),
               train_generator.classes)
return class_weights
```

Το κάθε convolution layer του δικτύου, όπως και το 1ο dense layer χρησιμοποιεί για συνάρτηση ενεργοποίησης την relu. Το τελευταίο layer χρησιμοποιεί φυσικά την softmax.

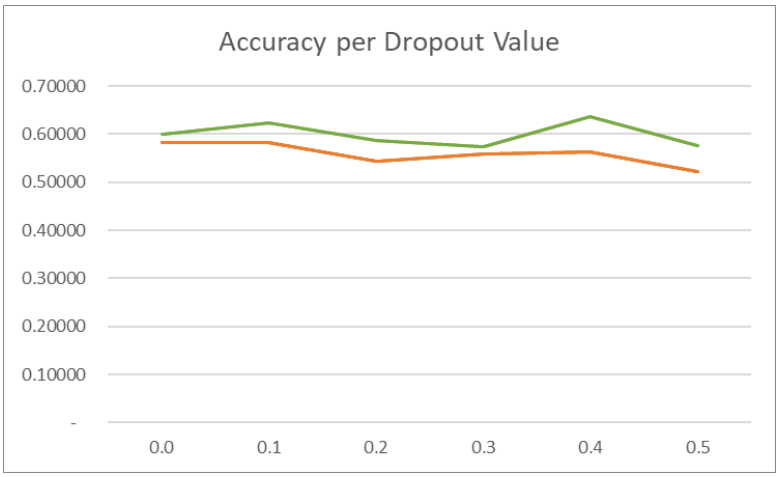
Ως optimizer δοκιμάστηκε κυρίως ο Adam. Ενώ δεν φάνηκε στις πρώτες δοκιμές ουσιαστική διαφορά ανάμεσα στον Adam και στον RMSprop, στις υπόλοιπες υλοποιήθηκαν δίκτυα μόνο με τον Adam για οικονομία χρόνου. Το learning rate, ενώ στις αρχικές δοκιμές ήταν σταθερό στο 0.0001, στις τελευταίες υλοποιήσεις (και έπειτα από αρκετές διακυμάνσεις) έμεινε σταθερό στο 0.0005. Ελέγχθηκε επίσης

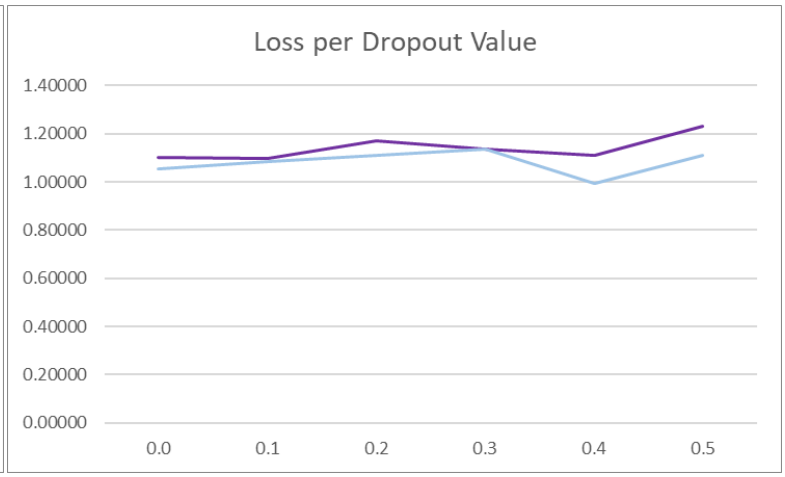
και ο SGD σε συνδυασμό με τον nesterov, χωρίς ικανοποιητικά αποτελέσματα όμως.

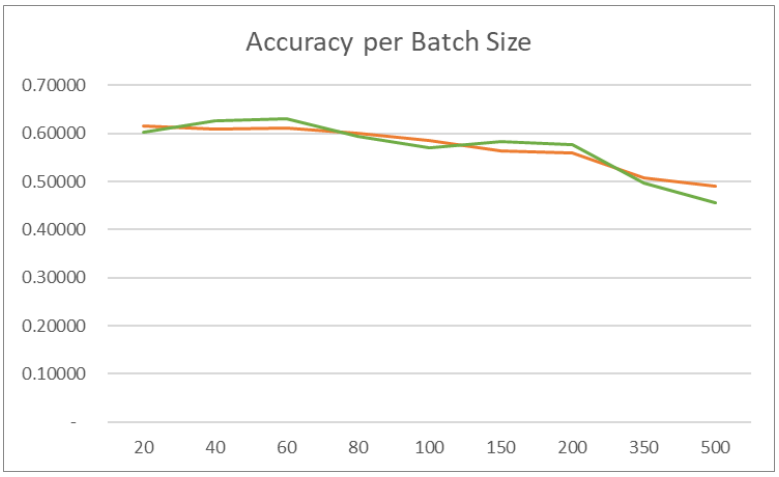
```
def build_model(learning_rate=0.0005,do=0.1):
    model = tf.keras.models.Sequential([
      tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
      tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
      tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
      tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
      tf.keras.layers.MaxPooling2D(pool_size=(6, 6)),

      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(128, activation='relu'),
      tf.keras.layers.Dropout(do),
      tf.keras.layers.Dense(6, activation='softmax')
    ])

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

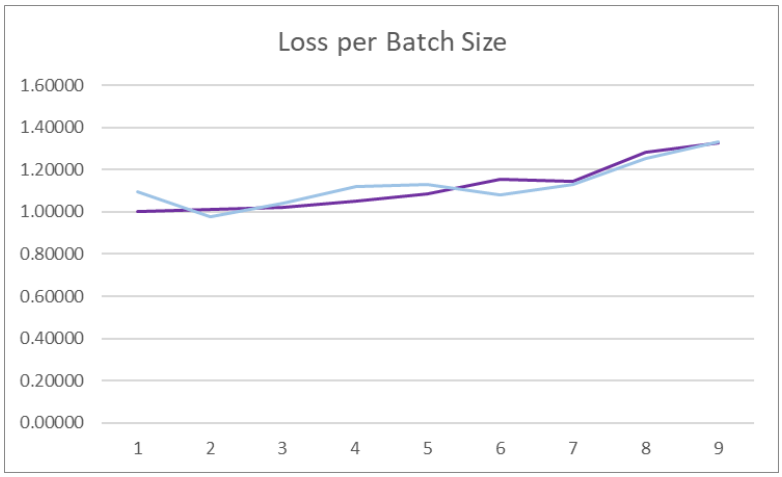
Διατηρώντας σταθερές όλες τις υπόλοιπες παραμέτρους, δοκιμάστηκε να προπονηθεί το δίκτυο για δέκα εποχές με πολλαπλές τιμές Dropout και batch_size, ώστε να επιλέγονταν μια ιδανική τιμή για το δίκτυο. Οι δοκιμές με τις διαφορετικές τιμές τους που έγιναν, έχουν καταγραφθεί στους παρακάτω πίνακες.

	Test		Validation		Parameters		
	Acc	Loss	Acc	Loss	Learning Rate	Dropout	Batch Size
1	0.58229	1.10295	0.60000	1.05577	0.00050	0.0	86
2	0.58229	1.09808	0.62333	1.08536	0.00050	0.1	86
3	0.54470	1.16911	0.58667	1.10947	0.00050	0.2	86
4	0.55890	1.13691	0.57333	1.13694	0.00050	0.3	86
5	0.56349	1.11171	0.63667	0.99456	0.00050	0.4	86
6	0.52297	1.23053	0.57667	1.11183	0.00050	0.5	86
7	0.61601	0.99979	0.60333	1.09379	0.00050	0.1	20
8	0.60799	1.00998	0.62667	0.97838	0.00050	0.1	40
9	0.61052	1.02151	0.63000	1.04374	0.00050	0.1	60
10	0.59983	1.05078	0.59333	1.11892	0.00050	0.1	80
11	0.58480	1.08768	0.57000	1.12828	0.00050	0.1	100
12	0.56442	1.15533	0.58333	1.08224	0.00050	0.1	150
13	0.56016	1.14434	0.57667	1.13113	0.00050	0.1	200
14	0.50746	1.28499	0.49667	1.25363	0.00050	0.1	350
15	0.48997	1.32660	0.45667	1.33471	0.00050	0.1	500









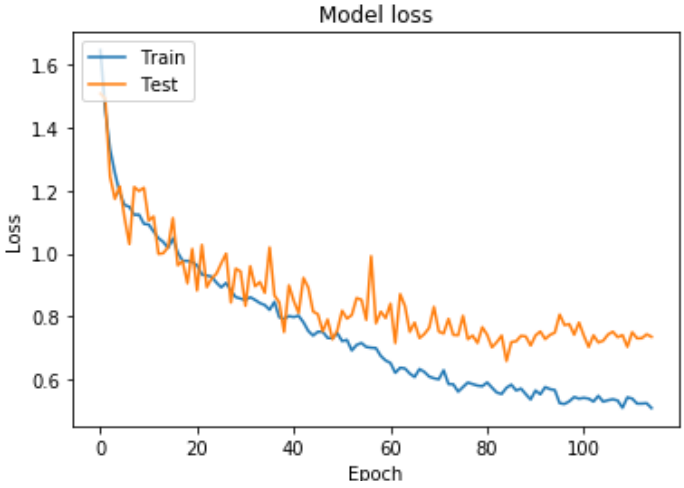
Παραδοχές:

- Το γεγονός και μόνο ότι σε κάθε εκπαίδευση το μοντέλο ξεκινάει με διαφορετικά βάρη, είναι αρκετό για να ορίσει την μέθοδο αξιολόγησης των παραμέτρων ως αναξιόπιστη, δεδομένου του μικρού αριθμού εποχών που δίνουμε στο μοντέλο για να εκπαιδευτεί.
- Μπορεί η αξιολόγηση να μην είναι εντελώς βάσιμη, αλλά μας δίνει τουλάχιστον ένα προβάδισμα ως προς το ποιες παραμέτρους θα επιλέγαμε για την τελική εκπαίδευση.

Ενδεικτικά αποτελέσματα μετά από την πλήρη εκπαίδευση των δικτύων με διαφορετικούς συνδυασμούς των παραπάνω παραμέτρων:

1) Dropout 0.1 , batch_size 60:

   Accuracy: 0.8069 | Loss: 0.5047
   Val. Accuracy: 0.7433 | Val. Loss: 0.7347
   Epochs: 115
   [Test Accuracy: 0.66773]

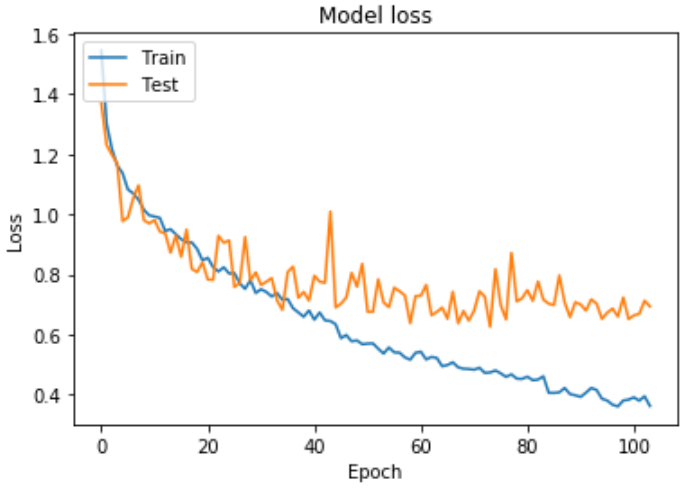


2) Dropout 0.1, batch_size 20:

   Accuracy: 0.8694 | Loss: 0.3628
   Val. Accuracy: 0.75 | Val. Loss: 0.6940
   Epochs: 104

   [Test Accuracy: 0.72523

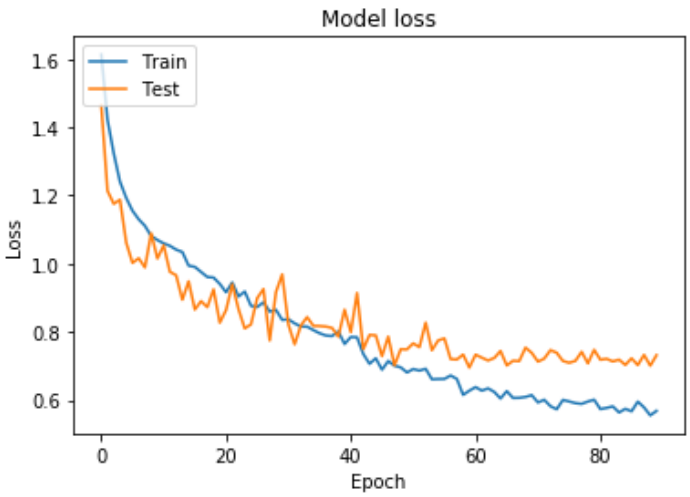
   0.74760 σε μια μεμονωμένη δοκιμή]



3) Dropout 0.4, batch_size 20:

   Accuracy: 0.7935 | Loss: 0.5690
   Val. Accuracy: 0.7467 | Val. Loss: 0.7319
   Epochs: 90

   [Test Accuracy: 0.72204]



Τα δύο μοντέλα που παραδόθηκαν στον διαγωνισμό, έχουν παραμέτρους Dropout 0.1, batch_size 20 και Dropout 0.4, batch_size 20 αντίστοιχα.

Πως εξηγούνται οι διακυμάνσεις στις εποχές εκπαίδευσης;

Η τελική εκπαίδευση του δικτύου περιλαμβάνει τα εξής:

```
callbacks = [tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss',factor=0.5,patience=10,verbose=1,mode='auto',cooldown=1),
             tf.keras.callbacks.EarlyStopping(monitor='val_loss',min_delta=0.001,patience=30,verbose=1,mode='auto')]

train_generator = create_train_gen(20)
class_weights = create_weights(train_generator)
model = build_model(0.0005,0.1)

history = model.fit_generator(generator=train_generator,
                    validation_data=validation_generator,
                    steps_per_epoch=train_generator.samples//train_generator.batch_size,
                    epochs=300,
                    verbose=1,
                    validation_steps=validation_generator.samples//validation_generator.batch_size,
                    callbacks=callbacks,
                    class_weight=class_weights)
```

ReduceLROnPlateau: Μικραίνει το learning rate του optimizer όταν η τιμή που παρατηρεί (σε εμάς το validation loss) δεν βελτιώνεται. Αποσκοπεί στο να τελειοποιήσει την ακρίβεια του μοντέλου με πολύ μικρότερες αλλαγές στα βάρη του δικτύου, όταν σταματήσει να βελτιώνεται η ακρίβεια του.

EarlyStopping: Σταματάει την εκπαίδευση όταν η τιμή που παρατηρεί (σε εμάς πάλι το validation loss) δεν βελτιώνεται. Έχει φυσικά μεγαλύτερη ανοχή από το ReduceLROnPlateau.

Class_weight: Κάθε κλάση έχει το δικό της βάρος (υπολογίστηκε στην αρχή με βάση το πόσο unbalanced είναι τα δεδομένα μας). Ανάλογα με το βάρος, η λάθος ταξινόμηση μιας εικόνας προσδίδει διαφορετικό loss.

Validation_data: Οι εικόνες που παίρνουμε από τον validation image generator για το validation set. Σε αντίθεση με τον train image generator, το batch_size του μένει σταθερό στα 100. Αυτό σημαίνει ότι ακόμη και αν σε κάθε βήμα της εποχής το δίκτυο προπονείται σε 20 εικόνες (σε περίπτωση που το batch size είναι 20), το validation γινεται σε 100. Έτσι έχουμε ακριβέστερη εικόνα για την ακρίβεια και (κυρίως) το loss του δικτύου στο κάθε step.

Epochs: Έχουν τεθεί αυθαίρετα σε πολύ μεγάλο αριθμό (300), ώστε να σταματήσει να προπονείται όταν κριθεί από το EarlyStopping ότι δεν υπάρχει βελτίωση της ακρίβειας του μοντέλου.

Αυτές ήταν οι παράμετροι με τις οποίες εκπαιδεύτηκαν τα δίκτυα που παραδόθηκαν στον διαγωνισμό.

Οι εικόνες του test set ταξινομήθηκαν και αποθηκεύτηκαν με τον εξής τρόπο:

```
results = []
test_images = os.listdir(test_dir)
for imgs in test_images:
    path = test_dir + "/" + imgs
    image = tf.keras.preprocessing.image.load_img(path, target_size=(150, 150), grayscale=False,
                                                  interpolation='nearest')
    image = tf.keras.preprocessing.image.img_to_array(image)
    image = image / 255.0
    image = np.expand_dims(image, axis=0)

    classes_pred = model.predict(image)
    results.append(np.argmax(classes_pred))

import pandas as pd

df = pd.DataFrame(list(zip(test_images, results)), columns =['Id', 'Category'])
df.to_csv('/kaggle/working/submission.csv', index=False)
df.head(60)
```

Αξίζει να σημειωθεί ότι και αυτές οι εικόνες κανονικοποιήθηκαν πριν κατηγοριοποιηθούν από το μοντέλο.

Στον διαγωνισμό δεν υποβλήθηκε pre-trained μοντέλο, λόγω της εκτενούς χρήσης του gpu time που μας έδινε το Kaggle τις τελευταίες τρεις εβδομάδες για την βελτίωση του παραπάνω μοντέλου.

Kaggle Username: Alex Karas (https://www.kaggle.com/ind66d)

Καρασακαλίδης Αλέξανδρος Ηλίας

19/01/2020