TECHNICAL REPORT – PORN DETECTION USING DEEP LEARNING

# Εισαγωγή

Υλοποιήθηκε ένα πρόγραμμα ανάλυσης εικόνων μεταξύ Porn και Bikini, χρησιμοποιώντας Deep Learning.

# Datasets

Ως Porn Dataset χρησιμοποιήθηκε από το Kaggle:

<https://www.kaggle.com/ljlr34449/porn-data/data>

Υλοποιήθηκε αλγόριθμος που διαβάζει σειριακά όλα τα urls των εικόνων του παραπάνω dataset για να γίνει η λήψη όλων των εικόνων.

*(320 x 180 η ανάλυση κάθε εικόνας)*

*(192.462 εικόνες)*

Για το Bikini Dataset δημιουργήθηκε με βάση τα αποτελέσματα εικόνων του Google search images, χρησιμοποιώντας το παρακάτω python script.

<https://github.com/hardikvasa/google-images-download>

Το Google search επιστρέφει περιορισμένο αριθμό εικόνων ανά αναζήτηση (600-800 εικόνες / query). Γι’ αυτό τον λόγο χρησιμοποιήθηκαν 54 διαφορετικά keywords για να γίνει επαρκής λήψη bikini εικόνων.

*“bathing bikini,  
 bathing suits,  
 beachbikini,  
 beachbikini hot,  
 beach bra,  
 beautiful bikini,  
 bikini,  
 bikinibeach,  
 bikiniexotic,*

*bikinihot,*

*bikinis,*

*…"*

27.426 εικόνες σύνολο

# Image Process

Δημιουργήθηκαν scripts τα οποία αφαιρούν corrupted και duplicate εικόνες για την καλύτερη και ορθή ανάλυση των όλων των εικόνων.

Όλες οι κατεβασμένες εικόνες από το google μετατράπηκαν σε 320 x 180 ίδιας διάστασης με τις εικόνες porn.

# Load Dataset

Δημιουργήθηκε μέθοδος load\_data η οποία φορτώνει όλες τις εικόνες από όλους τους φακέλους – υποφακέλους καθώς δημιουργεί και τα απαραίτητα labels.

Κατά την φόρτωση των εικόνων γίνεται resize στο μέγεθος 128x128 και αποθήκευση των features σε .h5 δεδομένα στον φάκελο *“load\_stuff”.*

Σε περίπτωση που υπάρχουν τα αρχεία: *“features.h5*” & *“labels.h5”* γίνεται φόρτωση των αρχείων για την χρήση τους. Γίνεται κανονικοποίηση των εικόνων σε τιμές 0 – 1 ( /255).

# Model

Χρησιμοποιήθηκε το παρακάτω νευρωτικό δίκτυο

1. model = Sequential()
2. model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=input\_shape))
3. model.add(Conv2D(32, (3, 3), activation='relu'))
4. model.add(MaxPooling2D(pool\_size=(2, 2)))
5. model.add(Dropout(0.25))
7. model.add(Conv2D(64, (3, 3), activation='relu'))
8. model.add(Conv2D(64, (3, 3), activation='relu'))
9. model.add(MaxPooling2D(pool\_size=(2, 2)))
10. model.add(Dropout(0.25))
12. model.add(Flatten())
13. model.add(Dense(256, activation='relu'))
14. model.add(Dropout(0.5))
15. model.add(Dense(2, activation='softmax'))
16. model.compile(loss='mse', optimizer='adam', metrics=[])
17. model.summary()
19. model.compile(loss='categorical\_crossentropy', optimizer='rmsprop', metrics=["accuracy"])

22. **print**("Training...")
23. model.fit(X\_train, y\_train, batch\_size=128, epochs=num\_epoch, validation\_data=(X\_test, y\_test), shuffle=True, verbose=2)
25. **print**("Test accuracy = %3.3f" % (model.evaluate(X\_test, y\_test)[1]))
27. model.save(model\_path)
28. **print**("[STATUS] saved model and weights to disk: ", model\_path)

# RUNTIME

1. Using TensorFlow backend.
2. Train length:  12426  Test length:  3107
3. input shape (128, 128, 3)
4. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
5. Layer (type)                 Output Shape              Param #
6. =================================================================
7. conv2d\_1 (Conv2D)            (None, 126, 126, 32)      896
8. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
9. conv2d\_2 (Conv2D)            (None, 124, 124, 32)      9248
10. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
11. max\_pooling2d\_1 (MaxPooling2 (None, 62, 62, 32)        0
12. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
13. dropout\_1 (Dropout)          (None, 62, 62, 32)        0
14. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
15. conv2d\_3 (Conv2D)            (None, 60, 60, 64)        18496
16. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
17. conv2d\_4 (Conv2D)            (None, 58, 58, 64)        36928
18. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
19. max\_pooling2d\_2 (MaxPooling2 (None, 29, 29, 64)        0
20. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
21. dropout\_2 (Dropout)          (None, 29, 29, 64)        0
22. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
23. flatten\_1 (Flatten)          (None, 53824)             0
24. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
25. dense\_1 (Dense)              (None, 256)               13779200
26. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
27. dropout\_3 (Dropout)          (None, 256)               0
28. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
29. dense\_2 (Dense)              (None, 2)                 514
30. =================================================================
31. Total params: 13,845,282
32. Trainable params: 13,845,282
33. Non-trainable params: 0
34. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
35. Training...
36. Train on 12426 samples, validate on 3107 samples
37. 2018-06-06 12:15:27.435438: I T:\src\github\tensorflow\tensorflow\core\common\_runtime\gpu\gpu\_device.cc:1356] Found device 0 with properties:
38. name: GeForce GTX 1080 Ti major: 6 minor: 1 memoryClockRate(GHz): 1.683
39. pciBusID: 0000:01:00.0
40. totalMemory: 11.00GiB freeMemory: 9.09GiB
41. 2018-06-06 12:15:27.435990: I T:\src\github\tensorflow\tensorflow\core\common\_runtime\gpu\gpu\_device.cc:1435] Adding visible gpu devices: 0
42. 2018-06-06 12:15:28.422635: I T:\src\github\tensorflow\tensorflow\core\common\_runtime\gpu\gpu\_device.cc:923] Device interconnect StreamExecutor with strength 1 edge matrix:
43. 2018-06-06 12:15:28.422832: I T:\src\github\tensorflow\tensorflow\core\common\_runtime\gpu\gpu\_device.cc:929]      0
44. 2018-06-06 12:15:28.422959: I T:\src\github\tensorflow\tensorflow\core\common\_runtime\gpu\gpu\_device.cc:942] 0:   N
45. 2018-06-06 12:15:28.423332: I T:\src\github\tensorflow\tensorflow\core\common\_runtime\gpu\gpu\_device.cc:1053] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 8804 MB memory) -> physical GPU (device: 0, name: GeForce GTX 1080 Ti, pci bus id: 0000:01:00.0, compute capability: 6.1)
46. Epoch 1/20
47. - 18s - loss: 1.0350 - acc: 0.7475 - val\_loss: 0.1951 - val\_acc: 0.9263
48. Epoch 2/20
49. - 11s - loss: 0.3275 - acc: 0.8967 - val\_loss: 0.3501 - val\_acc: 0.8622
50. Epoch 3/20
51. - 11s - loss: 0.2019 - acc: 0.9276 - val\_loss: 0.1361 - val\_acc: 0.9565
52. Epoch 4/20
53. - 11s - loss: 0.2016 - acc: 0.9403 - val\_loss: 0.0910 - val\_acc: 0.9707
54. Epoch 5/20
55. - 11s - loss: 0.1626 - acc: 0.9454 - val\_loss: 0.1191 - val\_acc: 0.9598
56. Epoch 6/20
57. - 11s - loss: 0.1214 - acc: 0.9555 - val\_loss: 0.0757 - val\_acc: 0.9775
58. Epoch 7/20
59. - 11s - loss: 0.1202 - acc: 0.9577 - val\_loss: 0.2063 - val\_acc: 0.9166
60. Epoch 8/20
61. - 11s - loss: 0.1135 - acc: 0.9589 - val\_loss: 0.0835 - val\_acc: 0.9781
62. Epoch 9/20
63. - 11s - loss: 0.1064 - acc: 0.9659 - val\_loss: 0.0563 - val\_acc: 0.9826
64. Epoch 10/20
65. - 11s - loss: 0.0905 - acc: 0.9691 - val\_loss: 0.0639 - val\_acc: 0.9817
66. Epoch 11/20
67. - 11s - loss: 0.0911 - acc: 0.9705 - val\_loss: 0.0626 - val\_acc: 0.9823
68. Epoch 12/20
69. - 11s - loss: 0.0791 - acc: 0.9734 - val\_loss: 0.0670 - val\_acc: 0.9804
70. Epoch 13/20
71. - 11s - loss: 0.0705 - acc: 0.9771 - val\_loss: 0.0606 - val\_acc: 0.9823
72. Epoch 14/20
73. - 11s - loss: 0.0740 - acc: 0.9771 - val\_loss: 0.0754 - val\_acc: 0.9781
74. Epoch 15/20
75. - 11s - loss: 0.0761 - acc: 0.9751 - val\_loss: 0.0747 - val\_acc: 0.9833
76. Epoch 16/20
77. - 11s - loss: 0.0675 - acc: 0.9774 - val\_loss: 0.0845 - val\_acc: 0.9794
78. Epoch 17/20
79. - 11s - loss: 0.0685 - acc: 0.9807 - val\_loss: 0.5578 - val\_acc: 0.8732
80. Epoch 18/20
81. - 11s - loss: 0.0636 - acc: 0.9817 - val\_loss: 0.0608 - val\_acc: 0.9874
82. Epoch 19/20
83. - 11s - loss: 0.0597 - acc: 0.9813 - val\_loss: 0.2409 - val\_acc: 0.9408
84. Epoch 20/20
85. - 11s - loss: 0.0622 - acc: 0.9815 - val\_loss: 0.0620 - val\_acc: 0.9855
87. 32/3107 [..............................] - ETA: 4s
88. 192/3107 [>.............................] - ETA: 1s
89. 352/3107 [==>...........................] - ETA: 1s
90. 512/3107 [===>..........................] - ETA: 1s
91. 672/3107 [=====>........................] - ETA: 1s
92. 832/3107 [=======>......................] - ETA: 0s
93. 992/3107 [========>.....................] - ETA: 0s
94. 1152/3107 [==========>...................] - ETA: 0s
95. 1312/3107 [===========>..................] - ETA: 0s
96. 1472/3107 [=============>................] - ETA: 0s
97. 1632/3107 [==============>...............] - ETA: 0s
98. 1792/3107 [================>.............] - ETA: 0s
99. 1952/3107 [=================>............] - ETA: 0s
100. 2112/3107 [===================>..........] - ETA: 0s
101. 2272/3107 [====================>.........] - ETA: 0s
102. 2432/3107 [======================>.......] - ETA: 0s
103. 2592/3107 [========================>.....] - ETA: 0s
104. 2752/3107 [=========================>....] - ETA: 0s
105. 2912/3107 [===========================>..] - ETA: 0s
106. 3072/3107 [============================>.] - ETA: 0s
107. 3107/3107 [==============================] - 1s 381us/step
108. Test accuracy = 0.986
109. [STATUS] saved model **and** weights to disk:  load\_stuff/model.h5

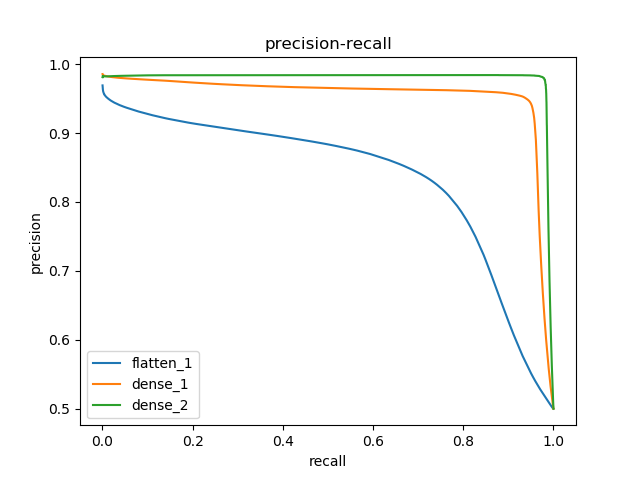
# METRICS

Το Precision είναι το ποσοστό των σωστών αποτελεσμάτων σε σχέση με το πλήθος των αποτελεσμάτων.

Το Recall είναι το ποσοστό των σωστών αποτελεσμάτων σε σχέση με το συνολικό αριθμό των σωστών δεδομένων που υπάρχουν στο dataset.

Έγινε μια μέτρηση precision / recall για τα 3 τελευταία layers του νευρωνικού δικτύου

*(Flatten 1, Dense 1, Dense 2)*



Παρατηρούμε πως το τελευταίο layer **Dense 2** έχει τα καλύτερα αποτελέσματα

# RESULTS - INFORMATION RETREIVAL

Χρησιμοποιήθηκε ο Nearest Neighbor για την εμφάνιση 30 παρόμοιων – κοντινών φωτογραφιών με βάση την είσοδο εικόνας.

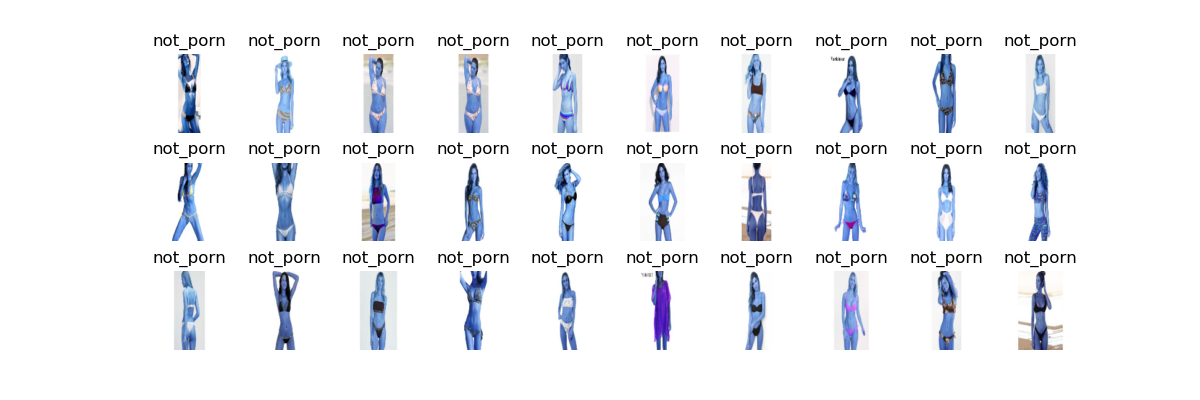
1. nn = NearestNeighbors(n\_neighbors=30, algorithm='brute', metric='cosine')
2. nn.fit(feat\_train)

Από το X\_test παίρνω τις πρώτες 2 εικόνες porn και not\_porn και εμφανίζω τις παρόμοιες εικόνες τις οποίες τις αποθηκεύω στον φάκελο ***results***.

Σε κάθε εικόνα εμφανίζω και ένα ποσοστό πιθανότητας μεταξύ “porn” / “notPorn”

1. proba = porn **if** porn > notPorn **else** notPorn
2. percent = proba \* 100

Εμφανίζω τις παρόμοιες εικόνες με βάση την εικόνα εισαγωγής και βάζοντας σε κάθε εικόνα την ετικέτα porn / not\_porn.







Πειραματικά χρησιμοποιήθηκε θόρυβος στις εικόνες εισόδου του predict στον Nearest Neighbor.

Συμπεραίνουμε πως στις εικόνες porn τις αναλύει αρκετά καλύτερα από τις εικόνες not\_porn.

Τα αποτελέσματα με salt\_n\_pepper βρίσκονται στον φάκελο ***results\normal\salt\_n\_pepper.***