Αναγνώριση Προτύπων

Τελική Εργασία

Ονοματεπώνυμο: Μωραΐτη Παναγιώτα

ΑΜ: 58054

Πίνακας περιεχομένων

Μέρος 1ο – Ταξινόμηση Εικόνων

Στο πρώτο μέρος θα επιλύσουμε ένα πρόβλημα ταξινόμησης εικόνων που προκύπτουν από αλγορίθμους ανίχνευσης προσώπων, με στόχο την επιβεβαίωση χρήσης μάσκας. Τα δεδομένα που χρησιμοποιούμε βρίσκονται στο αρχείο Mask_DB.zip και αποτελούνται από 1044 εικόνες προσώπων χωρίς τη χρήση μάσκας (without_mask), ισάριθμες εικόνες προσώπων που χρησιμοποιούν μάσκα (without_mask) και 56 εικόνες προσώπων που χρησιμοποιούν μάσκα εσφαλμένα (mask_incorrect_use).

Α. Διαχωρισμός δεδομένων

Αρχικά, χωρίζουμε τα δεδομένα τυχαία σε 60% δεδομένα εκπαίδευσης, 20% δεδομένα επικύρωσης και 20% δεδομένα δοκιμής, με τη χρήση της συνάρτησης **random_split** της pytorch. Στην ουσία, κρατάμε σε δυο λίστες τα ονόματα των εικόνων που βρίσκονται μέσα στους φακέλους without_mask και with_mask. Στη συνέχεια, για κάθε κατηγορία χωρίζουμε τις εικόνες στα 3 υποσύνολα που ζητούνται. Τέλος οι εικόνες από κάθε υποσύνολο αντιγράφονται σε 3 άλλους φακέλους. Οπότε πλέον έχουμε μέσα σε ένα νέο φάκελο, 3 φακέλους (train_dir, val_dir και test_dir). Μέσα σε καθέναν από αυτούς τους φακέλους έχουμε τους δύο φακέλους με τις εικόνες από τις δυο κλάσεις. Χρειαζόμαστε αυτή τη μορφή για να χρησιμοποιήσουμε τη συνάρτηση **ImageFolder** της Pytorch.

Η μορφή του dateset μας είναι πλέον η ακόλουθη:

-/split_dataset

--/train_dir

---/with_mask

---/without_mask

--/val_dir

---/with_mask

---/without_mask

-- /test_dir

---/with_mask

---/without_mask

Η συνάρτηση **ImageFolder** της Pytorch χρησιμοποιείται για να φορτωθούν οι εικόνες που βρίσκονται σε υποφακέλους, με κάθε υποφάκελο να αντιστοιχεί σε μια κατηγορία. Η συνάρτηση **ImageFolder** αναγνωρίζει αυτήν την δομή και αναθέτει αυτόματα μια ετικέτα (label) σε κάθε εικόνα, με βάση το όνομα του υποφακέλου στον οποίο βρίσκεται η εικόνα. Στη συνέχεια, μπορούμε να χρησιμοποιήσουμε την **ImageFolder** ως Dataset και σε συνδυασμό με έναν DataLoader μπορούμε να δώσουμε τις εικόνες σε ένα μοντέλο για εκπαίδευση και αξιολόγηση.

Στην παρακάτω εικόνα φαίνεται ο αριθμός των εικόνων που υπάρχουν σε κάθε υποσύνολο (υποφάκελο).

```
There are 3 directories and 0 images in '/content/Data Τελικής Εργασίας/Mask_DB/split_dataset'.
There are 2 directories and 0 images in '/content/Data Τελικής Εργασίας/Mask_DB/split_dataset/test_dir'.
There are 0 directories and 208 images in '/content/Data Τελικής Εργασίας/Mask_DB/split_dataset/test_dir/without_mask'.
There are 0 directories and 208 images in '/content/Data Τελικής Εργασίας/Mask_DB/split_dataset/test_dir/with_mask'.
There are 2 directories and 0 images in '/content/Data Τελικής Εργασίας/Mask_DB/split_dataset/train_dir'.
There are 0 directories and 627 images in '/content/Data Τελικής Εργασίας/Mask_DB/split_dataset/train_dir/without_mask'.
There are 0 directories and 627 images in '/content/Data Τελικής Εργασίας/Mask_DB/split_dataset/train_dir/with_mask'.
There are 2 directories and 0 images in '/content/Data Τελικής Εργασίας/Mask_DB/split_dataset/val_dir'.
There are 0 directories and 209 images in '/content/Data Τελικής Εργασίας/Mask_DB/split_dataset/val_dir/without_mask'.
There are 0 directories and 209 images in '/content/Data Τελικής Εργασίας/Mask_DB/split_dataset/val_dir/with_mask'.
```

Παρακάτω φαίνονται ενδεικτικά κάποιες εικόνες από το dataset και οι αντίστοιχες ετικέτες τους.

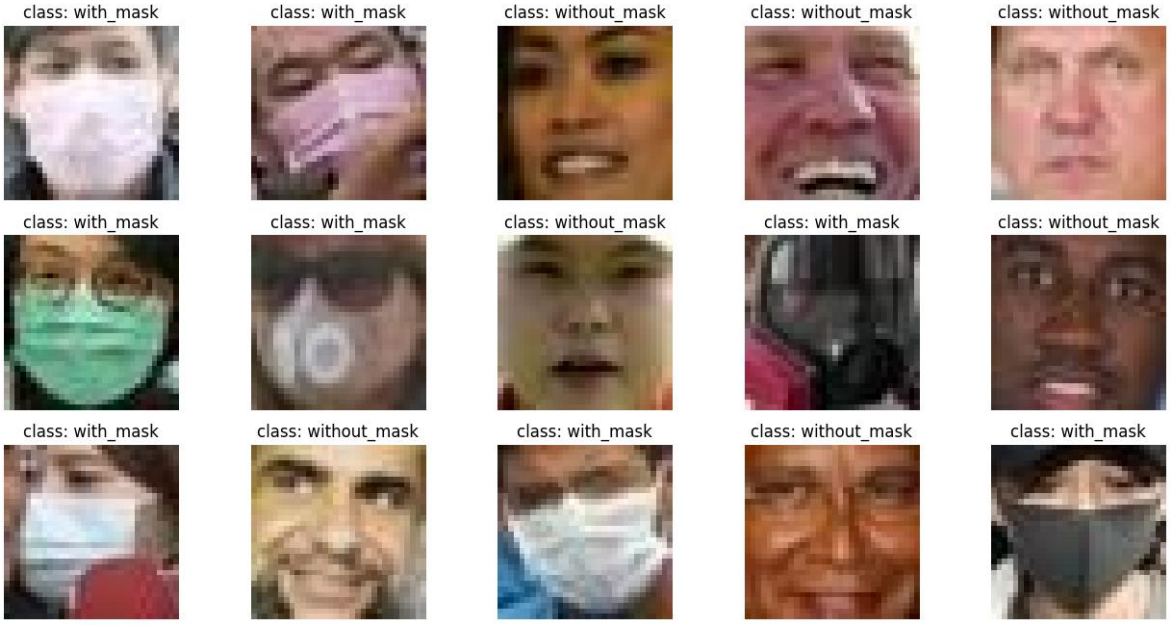
Επειδή, οι εικόνες που μας παρέχονται είναι σχετικά λίγες, θα δοκιμάσουμε να επαυξήσουμε το dataset με τη χρήση data augmentations, δηλαδή μικρών μετασχηματισμών στις εικόνες που διαθέτουμε. Με αυτόν τον τρόπο κάνουμε το μοντέλο πιο ανθεκτικό στο overfitting.

Πρέπει να προσέξουμε να μην κάνουμε πολύ επιθετικά augmentations, τα οποία μπορεί να είναι επιβλαβή στην απόδοση του δικτύου.

Επιλέξαμε να κάνουμε με τυχαίο τρόπο αναστροφή της εικόνας κατά μήκος του οριζόντιου άξονα (horizontal flip). Επίσης, δοκιμάσαμε να εφαρμόσουμε μετατόπιση και περιστροφή, τα οποία όμως αν και σε μικρό βαθμό φάνηκε να μην βοηθούν το δίκτυο, οπότε δεν διατηρήθηκαν.

```python
train_transform = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.RandomHorizontalFlip(),
    # transforms.RandomRotation(degrees=15), # Randomly rotate the image up to 15 degrees
    # transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)), # Randomly translate the image
    transforms.ToTensor()
])
```

Οι ίδιες εικόνες που παραθέσαμε παραπάνω φαίνονται μετά την εφαρμογή των augmentations.



Τα augmentations επιλέξαμε να τα εφαρμόσουμε μόνο στο training set (θα μπορούσαμε να τα εφαρμόσουμε και στο validation set).

Τέλος, φορτώνουμε τα δεδομένα σε batches με BATCH_SIZE 32. Επιλέγουμε να κάνουμε shuffling (ανακάτεμα των εικόνων), στην αρχή κάθε εποχής για όλα τα υποσύνολα (συνήθως για το test set δεν είναι απαραίτητο).

Παρακάτω φαίνονται τα mini batches που έχουν δημιουργηθεί.

```
Training set size: 1254
Validation set size: 418
Test set size: 416
```

```
Length of train dataloader: 40 batches of 32 images
Length of val dataloader: 14 batches of 32 images
Length of test dataloader: 13 batches of 32 images
```

Β. Σύστημα αναγνώρισης προτύπων για επιβεβαίωση χρήσης μάσκας

Επειδή το dataset αποτελείται από εικόνες, αποφασίσαμε να προχωρήσουμε στη χρήση κάποιου Συνελικτικού Νευρωνικού Δικτύου (Convolutional Neural Network). Τα CNNs έχουν σχεδιαστεί ειδικά για να χειρίζονται τα χαρακτηριστικά (features) εικόνων, λαμβάνοντας υπόψη την δομή των δεδομένων. Λαμβάνουν υπόψη τους τη θέση της πληροφορίας, η οποία είναι αυτή που καθιστά τα χαρακτηριστικά σε μια εικόνα. Επίσης, τα χαρακτηριστικά που ανιχνεύουν οργανώνονται σε μια ιεραρχία, από πολύ απλά features σε πολύ πιο σύνθετα. Όλα τα παραπάνω, αλλά και το γεγονός της επαναχρησιμοποίησης παραμέτρων (οι παράμετροι των φίλτρων επαναχρησιμοποιούνται μέσω της συνέλιξης, μείωση αριθμού παραμέτρων και ταχύτερη εκπαίδευση), καθιστούν τα CNNs κατάλληλα για την αναγνώριση προτύπων σε εικόνες, εξασφαλίζοντας υψηλή απόδοση στην ταξινόμηση, αλλά και σε άλλες εφαρμογές όρασης υπολογιστών.

Το δίκτυο που χρησιμοποιήθηκε αποτελείται από ένα επίπεδο conv2D (συνέλιξης) με 16 φίλτρα μεγέθους 3x3, συνάρτηση ενεργοποίησης ReLU, ένα επίπεδο Maxpooling

(υποδειγματοληψίας) με deafault παραμέτρους και ένα επίπεδο conv2D με 32 φίλτρα μεγέθους 3x3, συνάρτηση ενεργοποίησης ReLU, ένα επίπεδο Maxpooling. Στη συνέχεια, για να πάμε από τα επίπεδα συνέλιξης στα πλήρως συνδεδεμένα επίπεδα, έχουμε ένα επίπεδο flatten (μετατροπή 2D εξόδου σε διάνυσμα). Ο ταξινομητής (classifier) αποτελείται από 32 πλήρως συνδεδεμένους νευρώνες (fully connected), συνάρτηση ενεργοποίησης ReLU και 1 νευρώνα εξόδου.

Το μοντέλο και οι παράμετροι φαίνονται στην παρακάτω εικόνα.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
ModelV0                                  [32, 1]                   --
├─Sequential: 1-1                        [32, 32, 6, 6]            --
│    └─Conv2d: 2-1                       [32, 16, 30, 30]          448
│    └─ReLU: 2-2                         [32, 16, 30, 30]          --
│    └─MaxPool2d: 2-3                    [32, 16, 15, 15]          --
│    └─Conv2d: 2-4                       [32, 32, 13, 13]          4,640
│    └─ReLU: 2-5                         [32, 32, 13, 13]          --
│    └─MaxPool2d: 2-6                    [32, 32, 6, 6]            --
├─Sequential: 1-2                        [32, 1]                   --
│    └─Flatten: 2-7                      [32, 1152]                --
│    └─Linear: 2-8                       [32, 32]                  36,896
│    └─ReLU: 2-9                         [32, 32]                  --
│    └─Linear: 2-10                      [32, 1]                   33
==========================================================================================
Total params: 42,017
Trainable params: 42,017
Non-trainable params: 0
Total mult-adds (M): 39.18
==========================================================================================
Input size (MB): 0.39
Forward/backward pass size (MB): 5.08
Params size (MB): 0.17
Estimated Total Size (MB): 5.64
==========================================================================================
```

Επειδή έχουμε μόνο δυο κλάσεις (binary classification), μπορούμε να χρησιμοποιήσουμε σαν loss function τη συνάρτηση binary crossentropy. Στην έξοδο του δικτύου έχουμε έναν αριθμό, για να τον μετατρέψουμε σε πιθανότητα (από 0 έως 1) χρησιμοποιούμε τη συνάρτηση sigmoid. Στη συνέχεια, για να μετατρέψουμε την πιθανότητα σε πρόβλεψη, χρησιμοποιούμε ένα threshold 0.5 (πάνω από 0.5 έχει προβλεφθεί η κλάση 1). Αυτό μπορεί να γίνει με τη συνάρτηση round.

Ο optimizer που χρησιμοποιούμε είναι ο Adam.

Το learning rate ορίστηκε να είναι 0.001, όμως παρατηρήθηκε ότι το loss ανεβοκατέβαινε κατά την εκπαίδευση, οπότε μειώσαμε το ρυθμό μάθησης αρκετά. Οπότε, για να είναι πιο ομαλή η εκπαίδευση χρησιμοποιήθηκε learning rate scheduler και συγκεκριμένα StepLR, δηλαδή το lr ανά 5 εποχές γίνεται lr = lr * gamma, όπου το gamma το ορίσαμε να είναι 0.5. Άρα ανά 5 εποχές ο ρυθμός μάθησης μειώνεται στο μισό, κάνουμε μικρότερα βήματα καθώς εξελίσσεται η εκπαίδευση και πλησιάζουμε κάποιο ελάχιστο.

Τέλος, επιλέξαμε να εκπαιδεύσουμε για 150 εποχές με τη μέθοδο early stopping και patience 20 εποχές, δηλαδή αν δεν παρατηρηθεί βελτίωση (μείωση) στο validation loss για 20 εποχές σταματάει η εκπαίδευση.

Οι υπόλοιπες συναρτήσεις που χρησιμοποιήθηκαν για την εκπαίδευση και αξιολόγηση του δικτύου, καθώς και για τα διαγράμματα και τις προβλέψεις έχουν αναλυθεί και σε προηγούμενη εργασία, με τη διαφορά ότι εδώ τα αποτελέσματα αντί να αποθηκεύονται σε

ξεχωριστές λίστες, αποθηκεύονται σε λεξικά που περιέχουν σαν values λίστες, για την πιο εύκολη σύγκριση των διαφορετικών μοντέλων που θα μελετηθούν παρακάτω.

Οι παράμετροι των μοντέλων (state dictionary) αποθηκεύονται σε αρχεία .pth, ώστε αν λόγω τυχαιότητας πετύχουμε κάποιο πολύ καλό μοντέλο, να είμαστε σε θέση να το αναπαράγουμε. Επίσης, το καλύτερο μοντέλο μπορούμε να το κρατήσουμε, ώστε να μη χρειάζεται να ξαναεκπαιδεύουμε κάθε φορά που θέλουμε να το χρησιμοποιήσουμε, απλά φορτώνουμε τις βέλτιστες παραμέτρους που βρήκαμε.

Γ. Διαδικασία ρύθμισης παραμέτρων και αποτελέσματα

Αρχικά το μοντέλο που αναφέραμε παραπάνω και όλα τα μοντέλα που δοκιμάστηκαν, εκπαιδεύτηκαν και στα αρχικά δεδομένα και στα δεδομένα στα οποία έχουν εφαρμοστεί augmentations.

Ενδεικτικά στο notebook υπάρχουν ορισμένα μοντέλα που δοκιμάστηκαν, έγιναν και άλλες δοκιμές, αλλά εφόσον δεν οδήγησαν σε κάποιο καλύτερο αποτέλεσμα δεν συμπεριλήφθηκαν στο τελικό notebook. Τα αποτελέσματα των μοντέλων είναι τα ακόλουθα.

	model_name	train_loss	train_acc	val_loss	val_acc
0	model_0_without_augmentations	0.0648	97.89	0.0674	97.54
1	model_0_with_augmentations	0.0675	97.66	0.0662	97.77
2	model_1_without_augmentations	0.0031	100.00	0.0209	99.11
3	model_1_with_augmentations	0.0078	100.00	0.0281	98.66
4	model_2_without_augmentations	0.0040	100.00	0.0225	99.33
5	model_2_with_augmentations	0.0183	99.38	0.0298	99.11
6	model_3_without_augmentations	0.0040	100.00	0.0321	98.88
7	model_3_with_augmentations	0.0295	99.14	0.0428	98.44
8	model_4_without_augmentations	0.0051	100.00	0.0283	99.11
9	model_4_with_augmentations	0.0031	100.00	0.0336	98.88
10	model_5_without_augmentations	0.0005	100.00	0.0300	99.11
11	model_5_with_augmentations	0.0058	99.77	0.0201	99.33
12	model_6_without_augmentations	0.0075	99.84	0.0346	99.11
13	model_6_with_augmentations	0.0150	99.53	0.0328	99.11

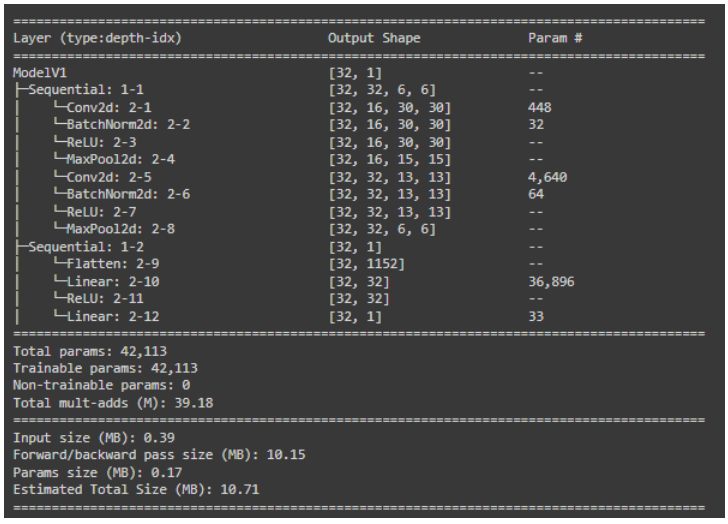
Το καλύτερο μοντέλο είναι:

	model_name	train_loss	train_acc	val_loss	val_acc
11	model_5_with_augmentations	0.0058	99.77	0.0201	99.33

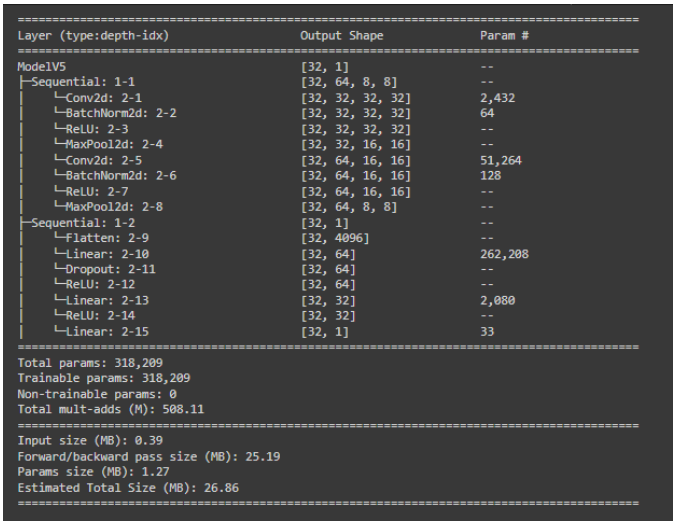
Αρχικά, παρατηρούμε ότι στα μοντέλα 1, 2, 3, 4, 6 τα augmentations φαίνεται να μην βοηθάνε τα μοντέλα και βλάπτουν και το val accuracy σε κάποιες περιπτώσεις. Αυτό ίσως

οφείλεται στο γεγονός ότι τα δίκτυα αυτά είναι μικρά σε μέγεθος. Για το μοντέλο 5 που έχουμε περισσότερες παραμέτρους, φαίνεται να βοηθάνε, σε πολύ μικρό βαθμό όμως.

Παρατηρούμε ότι η διαφορά στις παραμέτρους για το μοντέλο 5, σε σχέση με το μοντέλο 0, είναι πολύ μεγάλη.



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
ModelV1                                  [32, 1]                   --
├─Sequential: 1-1                        [32, 32, 6, 6]            --
│    └─Conv2d: 2-1                       [32, 16, 30, 30]          448
│    └─BatchNorm2d: 2-2                  [32, 16, 30, 30]          32
│    └─ReLU: 2-3                         [32, 16, 30, 30]          --
│    └─MaxPool2d: 2-4                    [32, 16, 15, 15]          --
│    └─Conv2d: 2-5                       [32, 32, 13, 13]          4,640
│    └─BatchNorm2d: 2-6                  [32, 32, 13, 13]          64
│    └─ReLU: 2-7                         [32, 32, 13, 13]          --
│    └─MaxPool2d: 2-8                    [32, 32, 6, 6]            --
├─Sequential: 1-2                        [32, 1]                   --
│    └─Flatten: 2-9                      [32, 1152]                --
│    └─Linear: 2-10                      [32, 32]                  36,896
│    └─ReLU: 2-11                        [32, 32]                  --
│    └─Linear: 2-12                      [32, 1]                   33
==========================================================================================
Total params: 42,113
Trainable params: 42,113
Non-trainable params: 0
Total mult-adds (M): 39.18
==========================================================================================
Input size (MB): 0.39
Forward/backward pass size (MB): 10.15
Params size (MB): 0.17
Estimated Total Size (MB): 10.71
==========================================================================================
```



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
ModelV5                                  [32, 1]                   --
├─Sequential: 1-1                        [32, 64, 8, 8]            --
│    └─Conv2d: 2-1                       [32, 32, 32, 32]          2,432
│    └─BatchNorm2d: 2-2                  [32, 32, 32, 32]          64
│    └─ReLU: 2-3                         [32, 32, 32, 32]          --
│    └─MaxPool2d: 2-4                    [32, 32, 16, 16]          --
│    └─Conv2d: 2-5                       [32, 64, 16, 16]          51,264
│    └─BatchNorm2d: 2-6                  [32, 64, 16, 16]          128
│    └─ReLU: 2-7                         [32, 64, 16, 16]          --
│    └─MaxPool2d: 2-8                    [32, 64, 8, 8]            --
├─Sequential: 1-2                        [32, 1]                   --
│    └─Flatten: 2-9                      [32, 4096]                --
│    └─Linear: 2-10                      [32, 64]                  262,208
│    └─Dropout: 2-11                     [32, 64]                  --
│    └─ReLU: 2-12                        [32, 64]                  --
│    └─Linear: 2-13                      [32, 32]                  2,080
│    └─ReLU: 2-14                        [32, 32]                  --
│    └─Linear: 2-15                      [32, 1]                   33
==========================================================================================
Total params: 318,209
Trainable params: 318,209
Non-trainable params: 0
Total mult-adds (M): 508.11
==========================================================================================
Input size (MB): 0.39
Forward/backward pass size (MB): 25.19
Params size (MB): 1.27
Estimated Total Size (MB): 26.86
==========================================================================================
```

Στο μοντέλο 1, επιλέξαμε να προσθέσουμε Batch Normalization layers μετά από κάθε conv2D layer. Τα Batch Normalization layers είναι μια μορφή κανονικοποίησης, που πολλές φορές συμβάλει στην βελτίωση της απόδοσης του δικτύου. Το val accuracy παρουσίασε σημαντική βελτίωση και το val loss μειώθηκε, αυτό σημαίνει ότι οι προβλέψεις που γίνονται είναι πιο confident και το συνολικό σφάλμα του δικτύου είναι μικρότερο από πριν. Οπότε, η τεχνική αυτή βοήθησε το δίκτυο και στα επόμενα μοντέλα θα χρησιμοποιηθεί.

Στο μοντέλο 2, σε κάθε conv2D layer χρησιμοποιήθηκε kernel size 5, αντί για 3 που είχαμε πριν. Ειδικά στο αποτέλεσμα με augmentations έχουμε αύξηση του val accuracy, αλλά αυξήθηκε ελάχιστα και το val loss (συνολικά μεγαλύτερο σφάλμα).

Στο μοντέλο 3, χρησιμοποιήθηκε padding, για τη διατήρηση των διαστάσεων μετά από τα επίπεδα συνέλιξης, κάτι που οδηγεί στη διατήρηση της πληροφορίας στο περιθώριο των εικόνων. Δε φάνηκε να προσφέρει σημαντική βελτίωση, αλλά ούτε και να είναι επιβλαβές για το δίκτυο. Γενικά είναι καλή τεχνική να χρησιμοποιούμε same padding όταν έχουμε πολλά επίπεδα συνέλιξης, γιατί κάποια στιγμή μπορεί να μειωθούν υπερβολικά οι διαστάσεις και να μην μπορούμε να προχωρήσουμε σε περαιτέρω συνελίξεις. Το kernel size εδώ είναι το αρχικό, δηλαδή 3.

Στο μοντέλο 4, χρησιμοποιήθηκε same padding και kernel size 5.

Στο μοντέλο 5, επιλέξαμε να αυξήσουμε τα φίλτρα στα δύο επίπεδα συνέλιξης που έχουμε. Αντί για 16 στο 1^{ο} επίπεδο και 32 στο 2^{ο}, δοκιμάσαμε να βάλουμε στο 1^{ο} επίπεδο 32 φίλτρα και στο 2^{ο} 64. Κρατήσαμε τα Batch Normalization layers, kernel size 5 και same padding. Επίσης, μεγαλώσαμε λίγο τον classifier, ενώ πριν είχαμε μόνο 32 νευρώνες και 1 νευρώνα εξόδου, τώρα βάλαμε 64 νευρώνες, ένα ακόμα επίπεδο με 32 νευρώνες και 1 νευρώνα εξόδου. Η αρχιτεκτονική αυτή φάνηκε να παρουσιάζει overfitting, με αποτέλεσμα την μειωμένη απόδοση του δικτύου. Οπότε, προσθέσαμε ένα Dropout layer με 0.25 πιθανότητα απενεργοποίησης κάθε νευρώνα, ανάμεσα στο επίπεδο με τους 64 και τους 32 νευρώνες. Το δίκτυο αυτό φάνηκε να επιτυγχάνει κάποια υψηλότερη απόδοση από πριν. Το val

accuracy έγινε **99.33%**, το οποίο το είχαμε επιτύχει και με προηγούμενο δίκτυο, όμως τώρα έχουμε πετύχει το ελάχιστο val loss **0.0201**. Αυτό σημαίνει ότι και το συνολικό σφάλμα είναι μικρότερο και οι προβλέψεις είναι σε πολύ μεγάλο βαθμό σωστές. Εδώ, τα augmentations βοήθησαν να το δίκτυο να επιτύχει καλύτερη γενίκευση, αφού χωρίς αυτά αν και το train accuracy είναι 100%, λόγω μικρού ποσοστού overfitting το val accuracy είναι 99.11%. Στη συνέχεια με τα augmentations το val accuracy αυξήθηκε ελάχιστα, παρόλο που το train accuracy είναι 99.77% (μικρότερο από πριν).

Στο μοντέλο 6 διατηρήσαμε τα αρχικά φίλτρα, 16 και 64, τα Batch Normalization layers, kernel size 5 και same padding. Στον classifier, επιλέξαμε πάλι δύο επίπεδα και το επίπεδο εξόδου, Το $1^{ο}$ επίπεδο έχει 32 νευρώνες και το $2^{ο}$ 16 νευρώνες. Το Dropout layer με 0.25 πιθανότητα απενεργοποίησης κάθε νευρώνα το κρατήσαμε. Αυτό το μοντέλο δε φάνηκε να επιτυγχάνει καλύτερα αποτελέσματα.

Το χαμηλότερο val loss (**0.0201**), το πέτυχε το μοντέλο 5, το οποίο έχει val accuracy **99.33%**.

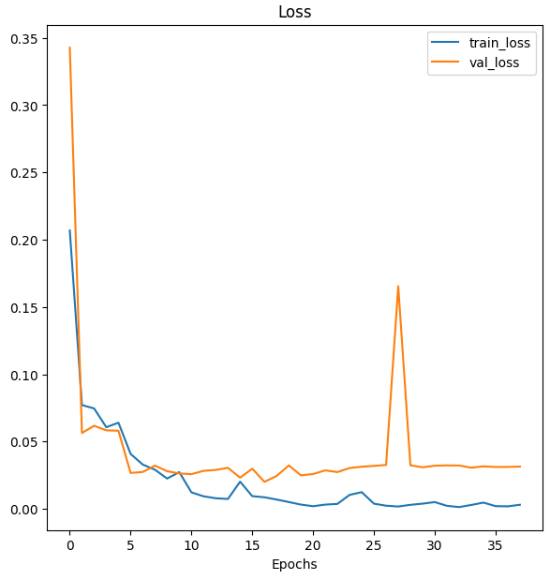
Η βελτίωση σε σχέση με το αρχικό μοντέλο είναι πάρα πολύ μικρή (μικρότερη από 2%). Αυτό σημαίνει ότι οι αλλαγές που κάναμε είχαν ελάχιστη επίδραση, επειδή δεν υπάρχουν μεγάλα περιθώρια βελτίωσης. Το δίκτυο που δοκιμάστηκε εξ αρχής ήταν πάρα πολύ καλό, για αυτό το πρόβλημα. Ειδικά για το δίκτυο 1 με τα υπόλοιπα, οι διαφορές στα αποτελέσματα είναι ελάχιστες.

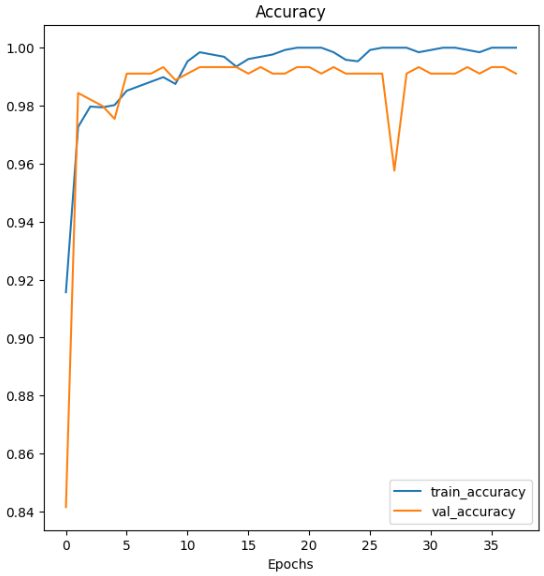
Δοκιμάσαμε μεγαλύτερα δίκτυα περισσότερα επίπεδα συνέλιξης και maxpooling και περισσότερα διαδοχικά επίπεδα συνέλιξης με λιγότερα επίπεδα maxpooling ενδιάμεσα), όμως η ακρίβεια στο val set σχεδόν πάντα ήταν μειωμένη, περίπου 96% με 97%. Στις περισσότερες περιπτώσεις αυτό οφειλόταν σε overffiting, καθώς στο training set, πετυχαίναμε 99% με 100% test accuracy. Ίσως κάποιο μεγαλύτερο δίκτυο με περισσότερα augmentations και Dropout Layers να τα πήγαινε καλύτερα, αν και τα δεδομένα είναι περιορισμένα οπότε καλό είναι να περιοριστούμε σε μικρότερα δίκτυα.

Οι διαφορές στα παραπάνω μοντέλα είναι πολύ μικρές και αν εκπαιδεύσουμε ξανά με διαφορετικά seeds (ακόμα με τα seeds που ορίσαμε δεν επιτυγχάνουμε κάθε φορά το ίδιο αποτέλεσμα, για απόλυτο reproducibility πρέπει να οριστούν επιπλέον seeds που δεν διερευνήθηκαν) μπορεί οι μικρές αλλαγές που θα υπάρξουν, να οδηγήσουν κάποιο άλλο μοντέλο στο καλύτερο αποτέλεσμα. Τα βάρη που οδηγούν στο καλύτερο αποτέλεσμα έχουν αποθηκευτεί για να μπορούμε να το αναπαράγουμε.

Κατά την εκπαίδευση, για την ανανέωση των βαρών χρησιμοποιούμε μόνο το training set. Το validation set χρησιμοποιείται για να παρακολουθούμε αν υπάρχει βελτίωση του validation loss/accuracy, αλλιώς να προχωρήσουμε σε early stopping. Το test set το χρησιμοποιούμε για την αξιολόγηση του βέλτιστου δικτύου, μόνο στο τέλος και όχι για τη ρύθμιση των παραμέτρων.

Τα αποτελέσματα για το βέλτιστο μοντέλο είναι τα ακόλουθα.






```
val_loss: 0.0201 | val_acc: 0.9933
```

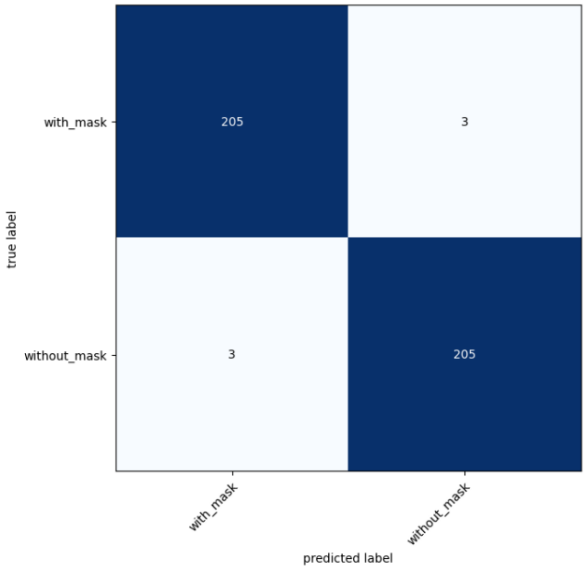
```
test_loss: 0.0692 | test_acc: 0.9856
```

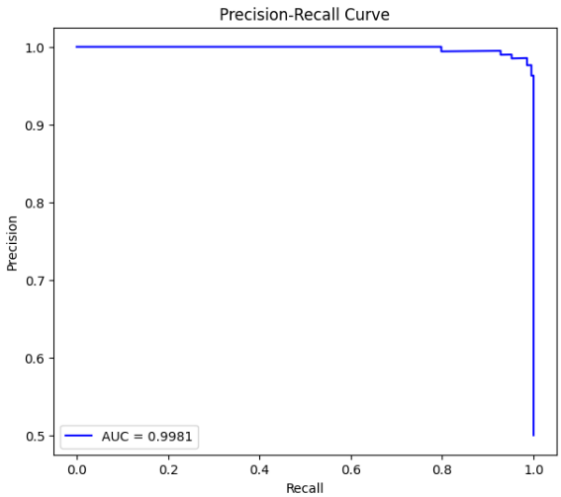
```
Classification accuracy of the best model in test set is: 98.5577%
Classification loss of the best model in test set is: 0.06922
```

```
Classification error of the best model in test set is: 0.01442
Images that are classified wrong are: 1.442%
```

Από τα διαγράμματα, παρατηρούμε ότι το validation loss ανεβαίνει σε ένα σημείο απότομα. Γενικά όμως η εκπαίδευση φαίνεται να εξελίσεται ομαλά, λόγω της κατάληλης ρύθμισης του learning rate.

Προβλέπουμε 3 εικόνες λάθος από κάθε κλάση.



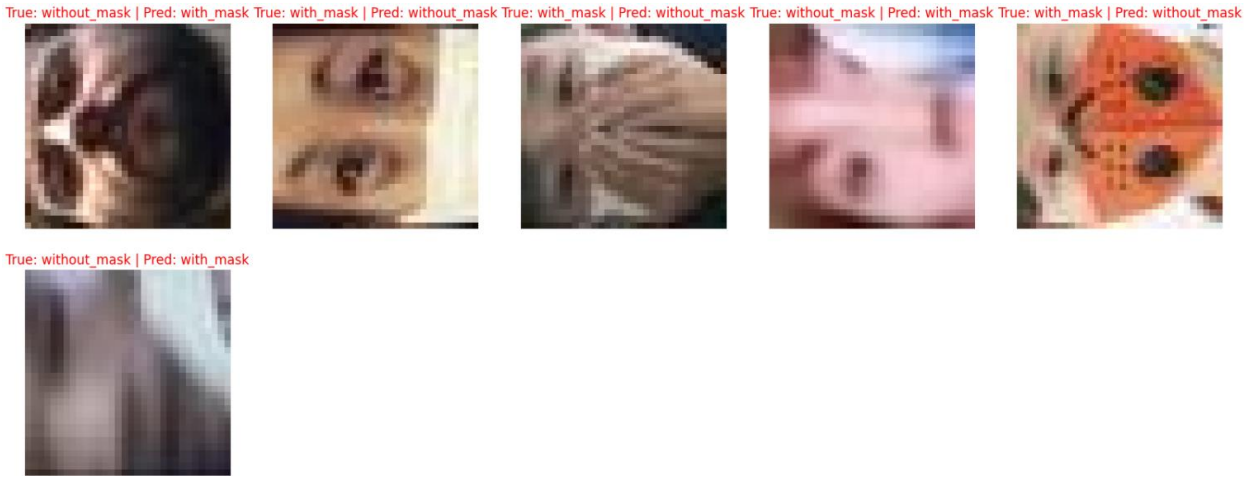


Το διάγραμμα precision-recall έγινε θεωρώντας ως θετική κλάση την κλάση without _mask.

```
{'with_mask': 0, 'without_mask': 1}
```

Το AUC είναι αρκετά κοντά στο 1, κάτι που υποδηλώνει ότι το μοντέλο έχει επιτύχει καλή διαχωρισιμότητα των κλάσεων.

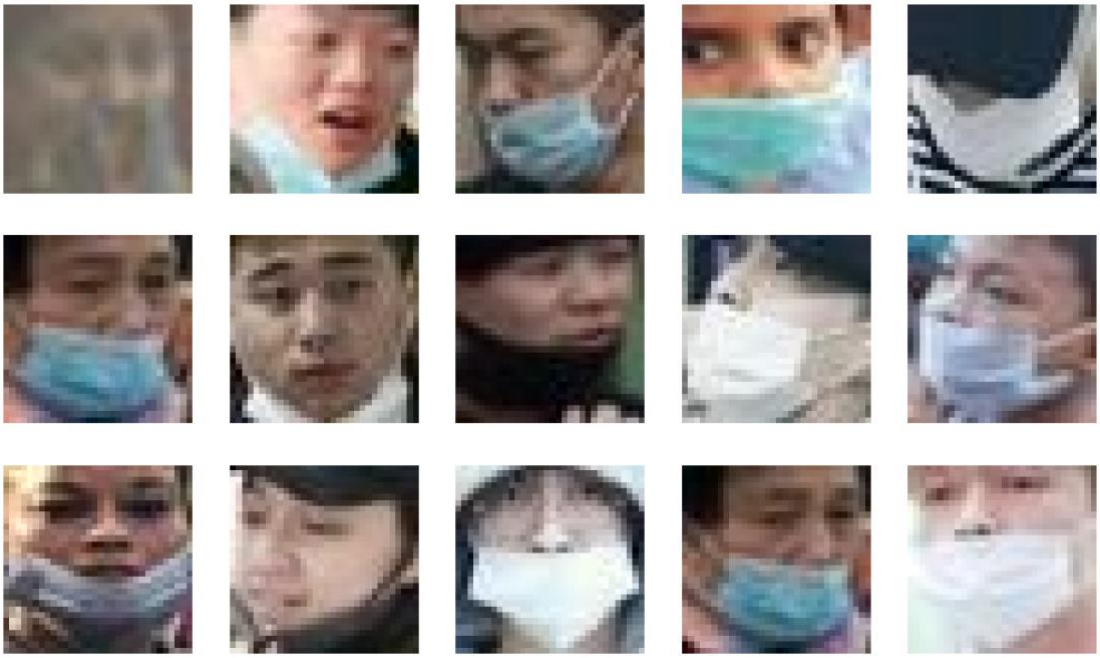
Οι εικόνες που προβλέφθηκαν λάθος είναι οι ακόλουθες.



Για την 6η εικόνα ακόμα και εμείς δεν μπορούμε να καταλάβουμε αν ο άνθρωπος φοράει μάσκα, καθώς η ποιότητά της είναι πάρα πολύ κακή. Οι υπόλοιπες 5 ίσως επειδή είναι στραμμένες κατά 90ο, για αυτό μπορεί να ταξινομήθηκαν ότι δε φοράνε μάσκα. Μία λύση θα ήταν να προσθέσουμε περισσότερα augmentations περιστροφής, πρέπει όμως να προσέξουμε να μην βλάψουμε τη συνολική απόδοση του δικτύου.

Δ. Εφαρμογή συστήματος σε εικόνες προσώπων που χρησιμοποιούν μάσκα εσφαλμένα

Κάποιες εικόνες του συνόλου mask_incorrect_use είναι οι ακόλουθες.



Εφαρμόσαμε το καλύτερο μοντέλο στο σύνολο mask_incorrect_use και λάβαμε τα ακόλουθα αποτελέσματα (threshold=0.5, **ελαστικό σύστημα** εντοπισμού ατόμων χωρίς μάσκα).

```
Number of images without masks: 9
Number of images with masks: 47
```

```
Accuracy in mask incorrect use is: 16.0714%
Classification error in mask incorrect use is: 0.8393
Classification error in mask incorrect use is: 83.9286%
```

Προβλέπουμε σωστά μόλις 9 εικόνες από τις 56. Ίσως αυτό συμβαίνει γιατί το μοντέλο έχει εστιάσει περισσότερο στη ύπαρξη μάσκας στην εικόνα, παρά στη σωστή χρήση της.

```
{'with_mask': 0, 'without_mask': 1}
```

Μία πιθανή λύση θα ήταν να κατεβάσουμε το threshold (από 0.5). Χαμηλότερο threshold σημαίνει ότι περισσότερες εικόνες θα προβλέπουμε ότι ανήκουν στην κλάση without_mask (κλάση 1). Αυτό βέβαια για να λειτουργήσει προυποθέτει ότι οι πιθανότητες για τις εικόνες mask_incorrect_use θα έχουν τιμές κοντά στο 0.5 και λίγο χαμηλότερες (πχ 0.4 ή λίγο παραπάνω). Σε διαφορετική περίπτωση, αν το μοντέλο είναι πολύ σίγουρο ότι οι εικόνες ανήκουν στην κλάση with_mask, οι πιθανότητες θα είναι κοντά στο 0, οπότε με μείωση του threshold δεν θα παρατηρηθεί κάποια διαφορά.

Δυστυχώς, οι πιθανότητες για τις εικόνες mask_incorrect_use είναι κοντά στο 0 και 0.1. Αυτό έχει ως αποτέλεσμα με ένα threshold 0.3 ή 0.4 να μην υπάρχει διαφορά με πριν. Η μόνη διαφορά που παρατηρήσαμε ήταν όταν το threshold οριστεί στο 0.1. Μια τόσο χαμηλή τιμή όμως μπορεί να βλάψει την συνολική απόδοση του δικτύου.

Για threshold 0.1 έχουμε.

```
Number of images without masks: 10
Number of images with masks: 46
```

```
Accuracy in mask incorrect use is: 17.8571%
Classification error in mask incorrect use is: 0.8214
Classification error in mask incorrect use is: 82.1429%
```

Για το test set.

```
test_loss: 0.0692 | test_acc: 0.9832
```

Για μια πιο ακραία τιμή threshold 0.001 έχουμε.

```
Number of images without masks: 21
Number of images with masks: 35
```

```
Accuracy in mask incorrect use is: 37.5%
Classification error in mask incorrect use is: 0.625
Classification error in mask incorrect use is: 62.5%
```

Για το test set.

```
test_loss: 0.0692 | test_acc: 0.9519
```

Για threshold 0.00001 (**πολύ αυστηρό σύστημα**) έχουμε.

```
Number of images without masks: 41
Number of images with masks: 15
```

```
Accuracy in mask incorrect use is: 73.2143%
Classification error in mask incorrect use is: 0.2679
Classification error in mask incorrect use is: 26.7857%
```

Για το test set.

```
test_loss: 0.0692 | test_acc: 0.8005
```

Για threshold 1e-10 (**πάρα πολύ αυστηρό σύστημα**) έχουμε.

```
Number of images without masks: 56
Number of images with masks: 0
```

```
Accuracy in mask incorrect use is: 100.0%
Classification error in mask incorrect use is: 0.0
Classification error in mask incorrect use is: 0.0%
```

Για το test set. Η ταξινόμηση πλησιάζει το 50% accuracy (τυχαία ταξινόμηση σε binary προβλήματα).

```
test_loss: 0.0692 | test_acc: 0.5457
```

Παρατηρούμε ότι το threshold πρέπει να μειωθεί πάρα πολύ για να έχουμε στο σύνολο mask_incorrect_use ακρίβεια πάνω από 50%. Παρά τη ραγδαία μείωση η ακρίβεια στο test set γίνεται 80.05% που είναι αρκετά μειωμένη σε σχέση με πριν, αλλά είναι μια σχετικά καλή τιμή. Πρέπει να αποφασίσουμε τι ποσοστό ακρίβειας θέλουμε να επιτυγχάνουμε στο mask_incorrect_use σε συνδυασμό με το ποσοστό ακρίβειας που ήμαστε διατεθειμένοι να θυσιάσουμε από το test_set για να κρίνουμε αν αυτή η μέθοδος μπορεί να λειτουργήσει στο πρόβλημά μας και στην περίσταση στην οποία θέλουμε να χρησιμοποιήσουμε το σύστημα. Για παράδειγμα, σε ένα πολύ αυστηρό σύστημα θα κατεβάσουμε πάρα πολύ το threshold και δεν μας ενδιαφέρει αν προβλέπω ότι κάποιος δε φοράει μάσκα, ενώ στην πραγματικότητα φοράει, επειδή θα τον ελέγξω και θα δω ότι φοράει, δε θέλω όμως να χάσω κάποιον που τη φοράει λάθος. Σε ένα πιο ελαστικό σύστημα, με ενδιαφέρει κυρίως να δω ποιοι δε φοράνε καθόλου μάσκα και ίσως και να εντοπίσω κάποιους λίγους που τη

φοράνε εντελώς λάθος, για να μην χρειάζεται να ελέγχω κάθε πολίτη και να είμαι σίγουρος ότι όταν εντοπίσω κάποιον που δε φοράει μάσκα, δε θα έχω κάνει λάθος.

Μία άλλη λύση θα ήταν να χρησιμοποιήσουμε περισσότερες εικόνες χωρίς μάσκα, κατά την εκπαίδευση, ώστε το μοντέλο να γέρνει πιο πολύ στην κλάση without_mask. Και αυτή η τεχνική όμως μπορεί να βλάψει την απόδοση του δικτύου (σε κλάση with_mask στο split λιγότερες εικόνες από 0.6 σε training set τις επιπλέον εικόνες τις βάζουμε στο validation και test set).

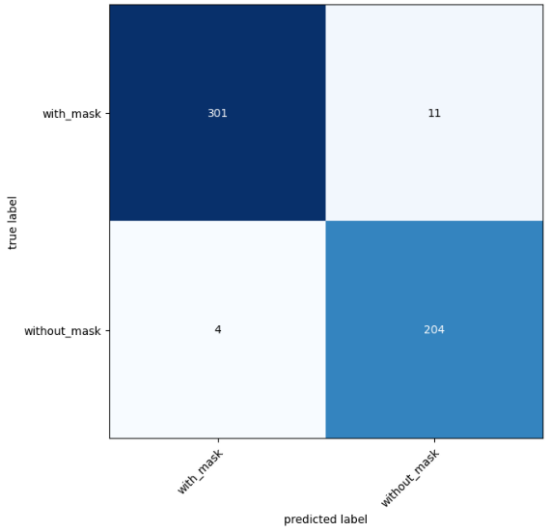
Έχουμε:

```
# Split percentages
train_per = 0.6 # 60%
val_per = 0.2 # 20%
test_per = 0.2 # 20%

train_per_with_mask = 0.4
val_per_with_mask = 0.3
test_per_with_mask = 0.3
```

Τα αποτελέσματα της εκπαίδευσης είναι:

```
test_loss: 0.0592 | test_acc: 0.9724
```



Στο σύνολο incorrect_mask_use:

```
Number of images without masks: 12
Number of images with masks: 44
```

```
Accuracy in mask incorrect use is: 21.4286%
Classification error in mask incorrect use is: 0.7857
Classification error in mask incorrect use is: 78.5714%
```

Οπότε, σε αντίθεση με το αρχικό δίκτυο που έβρισκε 9 εικόνες, τώρα βρίσκουμε 3 παραπάνω, δηλαδή 12 εικόνες. Βέβαια, όπως είναι αναμενόμενο το test accuracy μειώνεται στο 97.24% (από 98.56%)

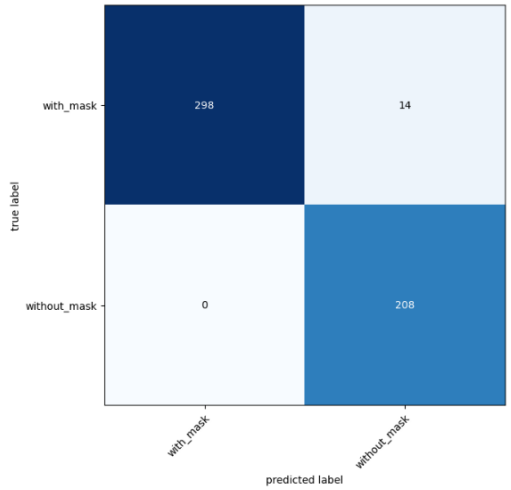
Με threshold 0.1 στο νέο δίκτυο για το σύνολο incorrect_mask_use:

```
Number of images without masks: 13
Number of images with masks: 43
```

```
Accuracy in mask incorrect use is: 23.2143%
Classification error in mask incorrect use is: 0.7679
Classification error in mask incorrect use is: 76.7857%
```

Στο test set:

```
test_loss: 0.0634 | test_acc: 0.9743
```



Έχουμε παραπάνω εικόνες σωστές, απότι στο αρχικό δίκτυο με threshold 0.1 (πριν ήταν 10 οι σωστές, τώρα 13). Μειώθηκε όμως πάλι το test accuracy στο 97.43% (πριν ήταν 98.32%).

Οπότε σε κάθε μία περίπτωση πρέπει να θυσιάσουμε σε κάποιο βαθμό την απόδοση του δικτύου για να καταφέρουμε να αναγνωρίζουμε ικανοποιητικό αιθμό εικόνων από το σύνολο mask_incorrect_use. Βέβαια το να εκπαιδεύουμε με λιγότερες εικόνες από κάποια κατηγορία μπορεί να έχει πιο απρόβλετη συμπεριφορά, οπότε αν μπορούσαμε να διαλέξουμε θα επιλέγαμε να μεταβάλουμε το threshold.

Επίσης, αντί να εκπαιδεύουμε με λιγότερα δεδομένα θα μπορούσαμε να δώσουμε μεγαλύτερη βαρύτητα, κατά την εκπαίδευση στα βάρη της κλάσης without_mask.

Προφανώς και αν μαζέψουμε περισσότερα δεδομένα με εικόνες που απεικονίζουν τη λανθασμένη χρήση μάσκας και τις ενσωματώσουμε στην κλάση without_mask, ίσως καταφέρουμε να κάνουμε το μοντέλο να εστιάζει στη χρήση της μάσκας και όχι στην ίδια τη ύπαρξη μάσκας. Επίσης θα μπορούσαμε να εκπαιδεύσουμε έναν autoencoder ή ένα generative model, για να παράγουμε τεχνητές εικόνες με λανθασμένη χρήση μάσκας και με αυτές να εκπαιδεύσουμε το δίκτυο (με 56 εικόνες αυτό είναι πολύ δύσκολο, αν είχαμε λίγες παραπάνω ίσως το καταφέρναμε).

Μέρος 2ο – Πρόβλεψη πρόσδεσης χημικών μορίων σε βιολογικούς υποδοχείς

Στο δεύτερο μέρος θα επιλύσουμε ένα πρόβλημα πρόβλεψης του βαθμού πρόσδεσης (binding) χημικών μορίων σε βιολογικούς υποδοχείς. Η διαδικασία αυτή χρησιμοποιείται συχνά στη φαρμακολογία και τη βιοχημεία για την ανακάλυψη χημικών μορίων που δυνητικά μπορούν να αποτελέσουν ενεργές ουσίες που προσδένονται σε κάποιο υποδοχέα-στόχο και αναστέλλουν τη δράση του. Στο αρχείο Data_Receptors.zip υπάρχουν δεδομένα εκπαίδευσης και δοκιμής για **1115** και 124 χημικά μόρια αντίστοιχα, για καθένα από τα οποία έχουν υπολογισθεί 3473 χαρακτηριστικά. Τα πρώτα **1425** χαρακτηριστικά (κατά στήλες) μπορούν να λάβουν συνεχείς τιμές, ενώ τα **2048** επόμενα αποτελούν binary περιγραφείς των μορίων. Για τα δεδομένα εκπαίδευσης υπάρχουν και οι ετικέτες που καθορίζουν εάν το κάθε μόριο μπορεί να προσδεθεί αποτελεσματικά στον υποδοχέα στόχο (label 1) ή όχι (label 0).

Α. Μοντέλο μηχανικής μάθησης για πρόβλεψη σύνδεσης μορίου σε υποδοχέα στόχο

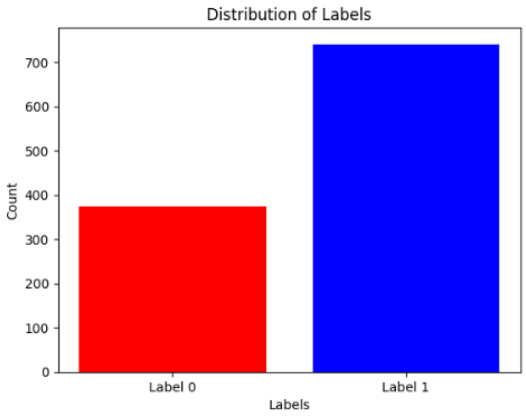
Παρατηρούμε ότι τα features είναι πολύ περισσότερα από τα δεδομένα που έχουμε στη διάθεση μας. Οπότε, λόγω του curse of dimensionality, είναι δύσκολο να χρησιμοποιήσουμε κάποιο νευρωνικό δίκτυο, καθώς τέτοια μοντέλα απαιτούν πάρα πολλά δεδομένα, διαφορετικά οδηγούμαστε σε overfitting.

Για τα δέντρα και τους ταξινομητές τύπου Random Forest, η επίδραση του curse of dimensionality είναι λιγότερο έντονη σε σύγκριση με πιο πολύπλοκα μοντέλα όπως τα νευρωνικά δίκτυα. Οι ταξινομητές αυτοί είναι σε θέση να διαχειριστούν δεδομένα για τα οποία ο αριθμός των features είναι συγκρίσιμος ή και μεγαλύτερος από τον αριθμό των δεδομένων στο dataset. Οπότε, θα επιλέξουμε να χρησιμοποιήσουμε κάποιο δεντρικό ταξινομητή/Random Forest ή κάποιο ensemble.

Τα ensemble μοντέλα έχουν τη δυνατότητα να αντιμετωπίσουν προβλήματα με υψηλή διαστατικότητα, καθώς συχνά εκπαιδεύονται σε διάφορα υποσύνολα των δεδομένων. Η συνδυαστική δύναμη των πολλαπλών μοντέλων βοηθά στη μείωση του overfitting και στη βελτίωση της γενίκευσης, ενώ παράλληλα επιτρέπει τη χρήση μεγαλύτερου αριθμού χαρακτηριστικών. Κάποια ensembles που επιλέξαμε να δοκιμάσουμε είναι το AdaBoost ή το XGBoost.

Αρχικά, αφαιρέσαμε τα features τα οποία έχουν την ίδια τιμή για όλα τα δείγματα (και για τα δείγματα στο training set και για τα δείγματα στο test set), αυτά τα χαρακτηριστικά δεν μας δίνουν κάποια πληροφορία. Από 3473 features, αφαιρέθηκαν τα 197 και μας έμειναν **3276** features.

Επειδή το dataset είναι unbalanced (διαφορετικός αριθμός δειγμάτων από κάθε κλάση), όταν εντοπίσουμε αρκετά υψηλό accuracy, θα κοιτάμε και τη μετρική F1 score ή το auc. Έχουμε 374 δείγματα από την κλάση 0 και 741 δείγματα από την κλάση 1.



Τα αποτελέσματα φαίνονται παρακάτω.

Decision Tree

```
Cross-validation scores: [0.87892377 0.89237668 0.90134529 0.91479821 0.85650224]
Mean CV accuracy: 88.8789%
```

```
Cross-validation F1 scores: [0.909699   0.91724138 0.9261745  0.93559322 0.89333333]
Mean CV F1 score: 0.9164
```

Random Forest

```
Best Parameters: {'max_depth': 10, 'n_estimators': 300}
```

```
Cross-validation scores: [0.88340807 0.91928251 0.92825112 0.93273543 0.89686099]
Mean CV accuracy: 91.2108%
```

```
Cross-validation F1 scores: [0.91447368 0.94117647 0.94805195 0.95114007 0.92698413]
Mean CV F1 score: 0.9364
```

AdaBoost

```
Cross-validation scores: [0.89237668 0.93273543 0.92376682 0.93721973 0.91479821]
Mean CV accuracy: 92.0179%
```

```
Cross-validation F1 scores: [0.91946309 0.95081967 0.94533762 0.95424837 0.93890675]
Mean CV F1 score: 0.9418
```

XGBoost

```
Best Parameters: {'max_depth': 6, 'n_estimators': 200}
```

```
Cross-validation scores: [0.88340807 0.94170404 0.93273543 0.94618834 0.92825112]
Mean CV accuracy: 92.6457%
```

```
Cross-validation F1 scores: [0.91333333 0.95709571 0.95049505 0.96       0.9483871 ]
Mean CV F1 score: 0.9459
```

Τα καλύτερα αποτελέσματα φαίνεται να τα δίνει ο XGBoost.

Β. Διαδικασία ρύθμισης παραμέτρων και εκτίμηση πιθανού σφάλματος σε σύνολο δοκιμής

Για την βελτιστοποίηση των παραμέτρων επιλέξαμε να χρησιμοποιήσουμε το πρωτόκολλο k-fold cross validation, το οποίο χωρίζει τα δεδομένα σε k folds και κάθε φορά χρησιμοποιεί ένα fold για testing και τα υπόλοιπα για training. Με αυτόν τον τρόπο εκμεταλλευόμαστε για την εκπαίδευση όλα τα δεδομένα και αξιολογούμε το μοντέλο μας σε διαφορετικά υποσύνολα.

Επίσης, για την διερεύνηση των βέλτιστων παραμέτρων, θα χρησιμοποιήσουμε την τεχνική GridSearch, η οποία διερευνά αυτόματα από ένα σύνολο παραμέτρων που θα τις δώσουμε ποιες είναι οι βέλτιστες.

Ο αλγόριθμος AdaBoost φαίνεται να είναι πολύ πιο αργός από τον XGBoost, οπότε με GridSearch αποφασίσαμε να διερευνήσουμε τις παραμέτρους μόνο για τον XGBoost, αλλά και για το Random Forest. Βέβαια, οι παράμετροι που μπορούμε να δοκιμάσουμε είναι περιορισμένοι, λόγω του αυξημένου χρόνου που χρειάζεται το GridSearch, καθώς οι παράμετροι αυξάνονται.

Τα αποτελέσματα με τις βέλτιστες παραμέτρους που διερευνήθηκαν είναι τα ακόλουθα και είναι βελτιωμένα σε σχέση με πριν.

Random Forest more hyperparameters tuning

```
# Random Forest
random_forest = RandomForestClassifier(n_estimators=350, max_depth=12, random_state=RANDOM_SEED)

# Perform cross-validation with 5 folds
cv_scores = cross_val_score(random_forest, X, y, cv=5)

# Print the cross-validation scores
print("Cross-validation scores:", cv_scores)

# Print the mean cross-validation score
print(f"Mean CV accuracy: {cv_scores.mean() * 100:.4f}%")
```

```
Cross-validation scores: [0.88340807 0.92376682 0.93721973 0.93721973 0.90134529]
Mean CV accuracy: 91.6592%
```

```
Cross-validation F1 scores: [0.91333333 0.9442623  0.95424837 0.95424837 0.92993631]
Mean CV F1 score: 0.9392
```

```
Mean AUC score: 0.97875
```

XGBoost more hyperparameters tuning

```
[ ]  # XGBoost
     xgboost = XGBClassifier(n_estimators=275, max_depth=6, random_state=RANDOM_SEED, n_jobs=-1)

     # Perform cross-validation with 5 folds
     cv_scores = cross_val_score(xgboost, X, y, cv=5)

     # Print the cross-validation scores
     print("Cross-validation scores:", cv_scores)

     # Print the mean cross-validation score
     print(f"Mean CV accuracy: {cv_scores.mean() * 100:.4f}%")
```

```
Cross-validation scores: [0.88340807 0.94618834 0.93273543 0.94618834 0.93273543]
Mean CV accuracy: 92.8251%
```

```
Cross-validation F1 scores: [0.91333333 0.9602649  0.95049505 0.96       0.95145631]
Mean CV F1 score: 0.9471
```

```
Mean AUC score: 0.98268
```

XGBoost less weight to class 1 (class with more samples)

Επειδή το dataset είναι unbalanced ίσως αν δώσουμε μεγαλύτερο βάρος στην κλάση με τα λιγότερα δείγματα κατά την εκπαίδευση, να βοηθήσουμε στη γενίκευση του ταξινομητή. Τα δείγματα της κλάσης 1 είναι διπλάσια από τα δείγματα της κλάσης 0. Βέβαια, αν δώσουμε το μισό βάρος στην θετική κλάση παρατηρείται μικρή μείωση στην ακρίβεια και το F1-score.

```
[ ]  # Define the XGBoost classifier
     xgboost = XGBClassifier(n_estimators=275, max_depth=6, scale_pos_weight=0.5, random_state=RANDOM_SEED, n_jobs=-1)

     # Define the StratifiedKFold
     stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_SEED)

     # Perform cross-validation with 5 folds using stratified k-fold
     cv_scores = cross_val_score(xgboost, X, y, cv=stratified_kfold)

     # Print the cross-validation scores
     print("Cross-validation scores:", cv_scores)

     # Print the mean cross-validation score
     print(f"Mean CV accuracy: {cv_scores.mean() * 100:.4f}%")

     Cross-validation scores: [0.9103139  0.92376682 0.93273543 0.92825112 0.91928251]
     Mean CV accuracy: 92.2870%
```

```
Cross-validation F1 scores: [0.93421053 0.94352159 0.94809689 0.94701987 0.94117647]
Mean CV F1 score: 0.9428
```

Αν το βάρος που θα δοθεί στην θετική κλάση είναι λίγο μικρότερο (95% του αρχικού), τότε παρατηρείται κάποια μικρή βελτίωση στην ακρίβεια και το F1-score. Βέβαια, το auc φαίνεται να είναι μειωμένο ελάχιστα.

```
# Define the XGBoost classifier
xgboost = XGBClassifier(n_estimators=275, max_depth=6, scale_pos_weight=0.95, random_state=RANDOM_SEED, n_jobs=-1)

# Define the StratifiedKFold
stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_SEED)

# Perform cross-validation with 5 folds using stratified k-fold
cv_scores = cross_val_score(xgboost, X, y, cv=stratified_kfold)

# Print the cross-validation scores
print("Cross-validation scores:", cv_scores)

# Print the mean cross-validation score
print(f"Mean CV accuracy: {cv_scores.mean() * 100:.4f}%")

Cross-validation scores: [0.91479821 0.93721973 0.94618834 0.92825112 0.91928251]
Mean CV accuracy: 92.9148%
```

```
Cross-validation F1 scores: [0.93811075 0.95364238 0.95945946 0.94771242 0.94117647]
Mean CV F1 score: 0.9480
```

```
Mean AUC score: 0.98080
```

Στη συνέχεια, θα μπορούσαμε να προχωρήσουμε σε μείωση διαστάσεων με τη χρήση PCA ή με κάποιον autoencoder, με την ελπίδα ότι η νέα αναπαράσταση σε χαμηλότερες διαστάσεις θα μας βοηθήσει να διαχωρίσουμε τις κλάσεις με μεγαλύτερη ακρίβεια.

Η PCA μπορεί να προχωρήσει σε μείωση διαστάσεων με τις μέγιστες διαστάσεις να είναι min(features, samples). Δηλαδή, οι μέγιστες διαστάσεις που μπορεί να μας επιστρέψει η PCA είναι όσα και τα δείγματα, 1115. Τα αποτελέσματα, παρόλα αυτά είναι τα ακόλουθα.

PCA 260 components

```
To explain 90% of the variance we need at least 47 principal components.
To explain 99% of the variance we need at least 260 principal components.
```

Random Forest

```
Cross-validation scores: [0.88340807 0.9103139  0.89686099 0.91479821 0.90134529]
Mean CV accuracy: 90.1345%
```

```
Cross-validation F1 scores: [0.91612903 0.93506494 0.92744479 0.93890675 0.93037975]
Mean CV F1 score: 0.9296
```

XGBoost

```
Cross-validation scores: [0.89686099 0.92376682 0.9058296  0.9058296  0.9058296 ]
Mean CV accuracy: 90.7623%
```

```
Cross-validation F1 scores: [0.92307692 0.94389439 0.93114754 0.93159609 0.93203883]
Mean CV F1 score: 0.9324
```

Κανονικοποίηση με StandarScaler (mean 0, variance 1), έγινε μόνο για τα συνεχή δεδομένα και όχι για τα binary. Οι βέλτιστες διαστάσεις επιλέχθηκαν ώστε να εξηγείται μεγάλο ποσοστό της αρχικής διακύμανσης (99%).

Στη συνέχεια, επιλέξαμε να διερευνήσουμε την περίπτωση που θα εφαρμόσουμε PCA μόνο στα συνεχή χαρακτηριστικά.

PCA 97 components μόνο στα συνεχή χαρακτηριστικά

```
To explain 90% of the variance we need at least 27 principal components.
To explain 99% of the variance we need at least 97 principal components.
```

Random Forest

```
Cross-validation scores: [0.91928251 0.93721973 0.92825112 0.9103139  0.9103139 ]
Mean CV accuracy: 92.1076%
```

```
Cross-validation F1 scores: [0.94       0.95394737 0.94805195 0.93548387 0.93630573]
Mean CV F1 score: 0.9428
```

XGBoost

```
Cross-validation scores: [0.87892377 0.92825112 0.92376682 0.9103139  0.91928251]
Mean CV accuracy: 91.2108%
```

```
Cross-validation F1 scores: [0.909699   0.94771242 0.94462541 0.93506494 0.94193548]
Mean CV F1 score: 0.9358
```

Με PCA επιτυγχάνουμε παρόμοια υψηλά ποσοστά επιτυχίας με πολύ λιγότερες διαστάσεις, τα οποία όμως δεν καταφέρνουν να ξεπεράσουν τα καλύτερά μας αποτελέσματα. Βέβαια,

με random forest τα αποτελέσματα είναι καλύτερα από αυτά που είχαμε πετύχει χωρίς μείωση διαστάσεων (91.659% ακρίβεια και 0.9392 F1-score είχαμε πριν).

Τέλος, υλοποιήθηκαν δύο autoencoders, ένας για τα συνεχή δεδομένα και ένας για τα binary. Αφού εκπαιδεύτηκαν, ώστε να δίνουν το ελάχιστο σφάλμα ανακατασκευής, διερευνήσαμε πόσο μπορούμε να μειώσουμε τις διαστάσεις ώστε το σφάλμα να παραμένει μικρό. Αν από τη συμπιεσμένη αναπαράσταση μπορούμε να ανακατασκευάσουμε τα δεδομένα, αυτό σημαίνει ότι πιθανώς οι διαστάσεις αυτές να είναι αρκετές για την εκπαίδευση των ταξινομητών.

Ο autoencoder για τα συνεχή χαρακτηριστικά δεν έδινε καθόλου καλά αποτελέσματα (ακόμα και 67% accuracy), οπότε δεν συμπεριλήφθηκε στον τελικό κώδικα.

Για τον autoencoder για τα binary χαρακτηριστικά επιλέγαμε να μειωθούν οι διαστάσεις από 2035 σε 1200 (κρατήσαμε σχεδόν τις μισές διαστάσεις).

```
Epoch [1/35], Loss: 0.0837
Epoch [2/35], Loss: 0.0782
Epoch [3/35], Loss: 0.0494
Epoch [4/35], Loss: 0.0599
Epoch [5/35], Loss: 0.0469
Epoch [6/35], Loss: 0.0331
Epoch [7/35], Loss: 0.0359
Epoch [8/35], Loss: 0.0371
Epoch [9/35], Loss: 0.0271
Epoch [10/35], Loss: 0.0267
Epoch [11/35], Loss: 0.0201
Epoch [12/35], Loss: 0.0218
Epoch [13/35], Loss: 0.0189
Epoch [14/35], Loss: 0.0184
Epoch [15/35], Loss: 0.0198
Epoch [16/35], Loss: 0.0177
Epoch [17/35], Loss: 0.0155
Epoch [18/35], Loss: 0.0121
Epoch [19/35], Loss: 0.0120
Epoch [20/35], Loss: 0.0119
Epoch [21/35], Loss: 0.0114
Epoch [22/35], Loss: 0.0090
Epoch [23/35], Loss: 0.0110
Epoch [24/35], Loss: 0.0106
Epoch [25/35], Loss: 0.0090
Epoch [26/35], Loss: 0.0088
Epoch [27/35], Loss: 0.0089
Epoch [28/35], Loss: 0.0102
Epoch [29/35], Loss: 0.0091
Epoch [30/35], Loss: 0.0107
Epoch [31/35], Loss: 0.0074
Epoch [32/35], Loss: 0.0090
Epoch [33/35], Loss: 0.0100
Epoch [34/35], Loss: 0.0113
Epoch [35/35], Loss: 0.0063
```

```
# Hyperparameters
input_size = 2035
latent_size = 1200
learning_rate = 0.001
num_epochs = 35
```

Random Forest

```
Cross-validation scores: [0.88340807 0.91928251 0.91479821 0.9058296  0.89686099]
Mean CV accuracy: 90.4036%
```

```
Cross-validation F1 scores: [0.91612903 0.94117647 0.93890675 0.93203883 0.92789969]
Mean CV F1 score: 0.9312
```

XGBoost

```
Cross-validation scores: [0.88340807 0.91928251 0.9058296  0.9103139  0.91479821]
Mean CV accuracy: 90.6726%
```

```
Cross-validation F1 scores: [0.91275168 0.94039735 0.93203883 0.93377483 0.93890675]
Mean CV F1 score: 0.9316
```

Δεν παρατηρείται βελτίωση στα αποτελέσματα.

Τέλος, με εφαρμογή και PCA στα συνεχή χαρακτηριστικά και autoencoder στα binary το accuracy και το F1-score μειώθηκαν.

Η μείωση διαστάσεων μόνο με PCA ή μόνο με autoencoder δίνει πολύ υψηλά ποσοστά επιτυχίας ακόμα και σε αρκετά χαμηλότερες διστάσεις. Ίσως με τη χρήση κάποιου άλλου ταξινομητή ή κάποιου νευρωνικού δικτύου, η απόδοση στις μειωμένες διαστάσεις να αυξανόταν σε μεγαλύτερο βαθμό. Οι ταξινομητές που βασίζονται σε decision trees πιθανότατα επειδή μπορούν να διαχειρίζονται πληθώρα χαρακτηριστικών, ίσως για αυτό να μην υπάρχει κάποια βελτίωση με τη μείωση διαστάσεων και είναι σε θέση εξ αρχής να αποδώσουν εξαιρετικά καλά.

Δοκιμάστηκαν επίσης διάφορα ensembles με τον XGBoost, όμως δεν υπήρξε κάποια βελτίωση.

<u>Τελικά αποτελέσματα συγκεντρωμένα</u>

	Accuracy	F1-Score	AUC
Decision Tree	88.8789%	0.9164	-
Random Forest	91.6592%	0.9392	0.97875
XGBoost	**92.8251%**	**0.9471**	**0.98268**
AdaBoost	92.0179%	0.9418	-

Για scale_pos_weight = 0.95

	Accuracy	F1-Score	AUC
XGBoost	**92.9148%**	**0.9480**	0.98080

Από 3473 χαρακτηριστικά αφαιρέσαμε τα χαρακτηριστικά τα οποία είχαν την ίδια τιμή για όλα τα δείγματα και έμειναν 3276.

Από 3276 διαστάσεις με PCA σε όλα τα χαρακτηριστικά πάμε σε **260** διαστάσεις.

Από 3276 διαστάσεις με PCA μόνο στα συνεχή χαρακτηριστικά πάμε σε **2132** διαστάσεις.

	PCA on all features		PCA only on continuous features	
	Accuracy	F1-Score	Accuracy	F1-Score
Random Forest	90.1345%	0.9296	**92.4664%**	**0.9452**
XGBoost	90.7623%	0.9324	91.2108%	0.9358

Από 3276 διαστάσεις με autoencoder μόνο στα binary χαρακτηριστικά πάμε σε **2441** διαστάσεις.

Με PCA μαζί με autoencoder πάμε σε **1297** διαστάσεις.

	Autoencoder on binary features		Autoencoder on binary features and PCA on continuous features	
	Accuracy	F1-Score	Accuracy	F1-Score
Random Forest	90.4036%	0.9312	79.6413%	0.8646
XGBoost	90.6726%	0.9316	88.3408%	0.9193

Το καλύτερο αποτέλεσμα μας το δίνει ο XGBoost. Βέβαια, για scale_pos_weight = 0.95 το accuracy και το F1-score αυξάνονται, όμως το auc μειώνεται ελάχιστα. Το auc υποδεικνύει υποδεικνύει καλύτερη διαχωριστική ικανότητα του ταξινομητή και σχετίζεται με το confidence για διαφορετικά thresholds. Οπότε, καλύτερα να επιλέξουμε τον ταξινομητή με το μεγαλύτερο auc που δείχνει καλύτερη γενίκευση. Βέβαια, η επιλογή εξαρτάται σε μεγάλο βαθμό από τις συγκεκριμένες ανάγκες και τα χαρακτηριστικά της εφαρμογής.

Επίσης, ο βέλτιστος ταξινομητής που επιλέξαμε δοκιμάστηκε και στα αρχικά features (πριν αφαιρεθούν τα features που έχουν την ίδια τιμή για όλα τα δείγματα). Τα αποτελέσματα για τις 3 μετρικές ήταν ακριβώς τα ίδια, κάτι που υποδεικνύει ότι η αφαίρεση αυτών των χαρακτηριστικών ούτε έβλαψε, αλλά και ούτε ωφέλησε τον ταξινομητή. Βέβαια, λιγότερα features οδηγούν και σε ταχύτερη εκπαίδευση οπότε αν κάποια χαρακτηριστικά δεν είναι χρήσιμα καλό είναι να αφαιρούνται. Επίσης, για το ερώτημα γ, παρατηρούμε ότι οι προβλέψεις και οι πιθανότητες είναι επίσης οι ίδιες είτε συμπεριλάβουμε όλα τα αρχικά features, είτε αγνοήσουμε τα features που έχουν την ίδια τιμή για όλα τα δείγματα. Οπότε, ο ταξινομητής μπορεί να καταλάβει από μόνος του ότι τα χαρακτηριστικά αυτά δεν είναι χρήσιμα και να τα αγνοήσει.

Περιμένουμε η απόδοση στο test set να είναι παρόμοια με τα αποτελέσματα του cross validation (στην ουσία γίνεται training-testing με διαφορετικά υποσύνολα του dataset). Οπότε, **περιμένουμε στο test set το πιθανό σφάλμα να είναι περίπου 7.1749%**.

```
Cross-validation scores: [0.88340807 0.94618834 0.93273543 0.94618834 0.93273543]
Mean CV accuracy: 92.8251%
```

Για διαφορετικά υποσύνολα του datset τα σφάλματα είναι τα ακόλουθα.

```
errors = [round((1-acc), 4) for acc in cv_scores]
errors
```

```
[0.1166, 0.0538, 0.0673, 0.0538, 0.0673]
```

```
mean_error = round(1-cv_scores.mean(), 4)
mean_error
```

```
0.0717
```

```
std_deviation = round(statistics.stdev(errors), 4)
std_deviation
```

```
0.026
```

Οπότε, το πιθανό σφάλμα στο test set θα να είναι περίπου $\mathbf{0.0717 \pm 0.026.}$

Επίσης, αξιολογήσαμε το μοντέλο και για 10 k-folds αντί για 5 (μέγεθος fold για 10 folds είναι περίπου 115 δείγματα). Τα αποτελέσματα ήταν πολύ κοντά (για 10 folds πιθανό σφάλμα $\mathbf{0.0789 \pm 0.0349}$).

Γ. Παραγωγή προβλέψεων σε σύνολο δοκιμής

Τέλος, εφόσον καταλήξαμε στον βέλτιστο ταξινομητή χρησιμοποιούμε **όλα τα δεδομένα για εκπαίδευση** (όσα περισσότερα δεδομένα έχω στη διάθεση μου τόσο καλύτερα θα αποδώσει ο ταξινομητής) και στη συνέχεια προχωράω στις προβλέψεις για κάθε δείγμα από το test set και αποθηκεύω τις προβλέψεις και τις πιθανότητες σε ένα αρχείο csv (1η στήλη το label που έχω προβλέψει και 2η στήλη η πιθανότητα πρόσδεσης).

Χρησιμοποιήσαμε όλα τα αρχικά features για την εκπαίδευση. Σε κάποιο άγνωστο test set στο οποίο δεν ελέγχουμε τα features, πιθανόν αν τα features που έχουν την ίδια τιμή για όλα τα δείγματα στο training set, έχουν διαφορετικές τιμές στο test set, ίσως να υπάρχει απρόβλεπτη συμπεριφορά και δεν γνωρίζουμε τελικά αν πρέπει να τα αγνοήσουμε ή να τα συμπεριλάβουμε στην εκπαίδευση. Πιθανότατα, αν ο ταξινομητής μάθει κατά την εκπαίδευση να αγνοεί αυτά τα χαρακτηριστικά, θα τα αγνοήσει και κατά το testing.

Πιθανότατα επειδή χρησιμοποιήσαμε όλα τα δεδομένα για εκπαίδευση, το σφάλμα ίσως είναι λίγο μικρότερο από αυτό που εκτιμήσαμε χρησιμοποιώντας ένα μέρος του συνόλου εκπαίδευσης για testing.