ΕΡΓΑΣΙΑ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ - ΑΝΑΦΟΡΑ

Καυκάς Πέτρος mtn2204

Περιεχόμενα

1. Περιγραφή προβλήματος

1^{α} – ποιοτικά στοιχεία

Σε αυτήν την εργασία ασχοληθήκαμε με το πρόβλημα της ταυτοποίησης εικόνας (image classification problem) και, γενικότερα, με τον τομέα του computer vision. Πιο συγκεκριμένα, επιλέξαμε να καταπιαστούμε με πίνακες ζωγραφικής και στοχεύσαμε να υλοποιήσουμε μία εφαρμογή αναγνώρισης/πρόβλεψης του καλλιτέχνη – ζωγράφου. Για αυτόν τον σκοπό, χρησιμοποιήσαμε πίνακες από τέσσερις (4) καλλιτέχνες, τους ζωγράφους: Πικάσο, Μονέ, Ρέμπραντ και Νταλί. Επιλέξαμε τους συγκεκριμένους ζωγράφους για τους εξής λόγους: Αφενός, και οι τέσσερις υπήρξαν πολύ καλλιτεχνικά παραγωγικοί στην ζωή τους και έτσι είχαμε στη διάθεσή μας ένα μεγάλο αριθμό πινάκων για καθέναν από αυτούς (>500). Αφετέρου, υπάγονται σε διαφορετικά καλλιτεχνικά ρεύματα, (αντιστοίχως με πριν: κυβισμός, ιμπρεσιονισμός, ρεαλισμός και σουρεαλισμός), ώστε έχει ένα ενδιαφέρον το να «μάθει» ένας υπολογιστής να τους ξεχωρίζει. Τέλος, παρότι ανήκουν σε διαφορετικά ρεύματα, η διαφορά στο στυλ τους είναι τόση ώστε να μην είναι αυτονόητα απλός και τετριμμένος ο διαχωρισμός τους, πράγμα που σημαίνει ότι υπάρχει και ένας υπολογίσιμος βαθμός δυσκολίας για να επιτευχθεί (ειδικά με χρήση αλγορίθμων παραδοσιακού Computer Vision, χωρίς να αγγίζουμε την βαθιά μηχανική μάθηση).

1^{β} – τεχνικά στοιχεία

Χρησιμοποιήσαμε ένα python script: https://github.com/lucasdavid/wikiart ώστε να «κατεβάσουμε» πίνακες από την σελίδα: https://www.wikiart.org/ για κάθε έναν από τους τέσσερις καλλιτέχνες μας. Από τους συνολικούς πίνακες που κατεβάσαμε χρησιμοποιήσαμε πεντακόσιους (500) για να εκπαιδεύσουμε τον αλγόριθμό μας, και περίπου εκατό (100) για σκοπούς επίδειξης (demo) και validation. Το εργαλείο που χρησιμοποιήσαμε για να υλοποιήσουμε τον classifier είναι το sklearn. Παρακάτω παραθέτουμε έναν πίνακα με τα βασικά στοιχεία του προβλήματος:

Καλλιτέχνης	Ρεύμα	Αριθμός Πινάκων
Pablo Picasso	Κυβισμός	500 (train/test-validation) + 100(demo/val)
Claude Monet	Ιμπρεσιονισμός	500 (train/test-validation) + 100(demo/val)
Harmenszoon Rembrandt	Ρεαλισμός	500 (train/test-validation) + 100(demo/val)
Salvador Dali	Σουρεαλισμός	500 (train/test-validation) + 100(demo/val)

Η Υλοποίηση που πραγματοποιήσαμε μπορεί να λειτουργήσει με οποιουσδήποτε τέσσερις καλλιτέχνες (ή και παραπάνω) αρκεί να τοποθετηθούν στους ανάλογος φακέλους του προγράμματος οι κατάλληλοι πίνακες. Τέλος, δεν χρησιμοποιήθηκαν καθόλου αλγόριθμοι deep learning, παρά μόνο βασιστήκαμε σε παραδοσιακούς αλγορίθμους του Computer Vision.

2. Περιγραφή της υλοποίησης

2^{α}. – Προετοιμασία δεδομένων.

Τα δεδομένα στην βασική τους μορφή είναι εικόνες της μορφής .jpg τις οποίες κατεβάσαμε χρησιμοποιώντας ένα python script. Οι διαστάσεις και οι αναλύσεις τους, συνεπώς, στην αρχή ήταν ανομοιογενείς. Για αυτόν τον λόγο στο αρχικό μέρος της υλοποίησής μας δημιουργήσαμε τις κατάλληλες functions ώστε να μετασχηματίσουμε τις εικόνες και να τις ομογενοποιήσουμε. Συγκεκριμένα, δοκιμάσαμε διάφορες διαστάσεις και φίλτρα χρώματος, και καταλήξαμε πως ο βέλτιστος συνδυασμός, δεδομένου του περιορισμού των υπολογιστικών πόρων που είχαμε, είναι εικόνες διαστάσεων 64x64 pixels, και σε γκρι κλίμακα χρώματος. Κάθε μία εικόνα λοιπόν, εφόσον την «διαβάζουμε», μετέπειτα μετασχηματίζουμε τις διαστάσεις τις και αλλάζουμε το φίλτρο χρώματός της, και τέλος την μετατρέπουμε σε numpy array. Αυτή η διαδικασία υλοποιείται από την συνάρτηση *data_process(img_path)* στον αλγόριθμό μας. Αποθηκεύουμε το αποτέλεσμά της στην array με το όνομα x_train και x_train_labels (η πρώτη έχει τα δεδομένα της εικόνας, ενώ η δεύτερη το όνομα κάθε εικόνας).

2^{β} – Feature Extraction

Κατόπιν της μετατροπής των εικόνων σε numpy arrays, χρησιμοποιούμε την συνάρτηση *feature_extractor(dataset)* ώστε να εξάγουμε τα features που μας ενδιαφέρουν για την κατηγοριοποίηση των εικόνων (image classification). Τα features που έχουμε επιλέξει να εξάγουμε για κάθε εικόνα είναι:

1. Τα pixel values της εικόνας νορμαλοποιημένα
2. Τέσσερα Gabor Filters σε κάθε εικόνα
3. Τα hog features κάθε εικόνας

Οι τιμές των παραπάνω features, εφόσον εξαχθούν, καταχωρούνται σε ένα pandas dataframe με την μορφή του παρακάτω πίνακα:

Pixel Values	Gabor1	Gabor2	Gabor3	Gabor4	Hog
…	…	…	…	…	…

Κάθε στήλη περιέχει το ομώνυμο feature για κάθε εικόνα. Αποθηκεύουμε αυτό το dataset με το όνομα *image_features*.

Έπειτα, χρησιμοποιούμε την επόμενη function, create_feature_matrix(x_train, image_features) ώστε να δημιουργήσουμε το feature vector για κάθε εικόνα. Το τελικό αποτέλεσμα αποθηκεύεται στο pandas dataframe με το όνομα feature_matrix. Το dataframe αυτό έχει την εξής μορφή:

1	2	3	4	5	…	24280	Painter
…	<feature values>						<painter name>
2000							

Συνολικά λοιπόν έχουμε δύο χιλιάδες γραμμές (μία για κάθε έναν από του πίνακές μας) και κάθε γραμμή περιέχει τα pixel values, τα gabor filters και τα hog στην σειρά, το ένα μετά το άλλο: 64 pixel επί 64 pixel = 4096 values. Αυτά τα 4096 values για κάθε pixel τα έχουμε πέντε φορές γιατί έχουμε την αρχική εικόνα συν τέσσερα αντίγραφα για κάθε ένα gabor filter συν 3800 στοιχεία για τα hog features. Συνεπώς συνολικά έχουμε 4096x5 + 3800 = 24280 στήλες. Παρακάτω παραθέτουμε μία πρώτη «εικόνα» του προβλήματος, εφόσον περάσαμε τον παραπάνω πίνακα από μία διαδικασία απομείωσης των διαστάσεων σε τρεις (3) με την χρήση της μεθόδου PCA, ώστε να μπορούμε να έχουμε μία οπτική αναπαράσταση.

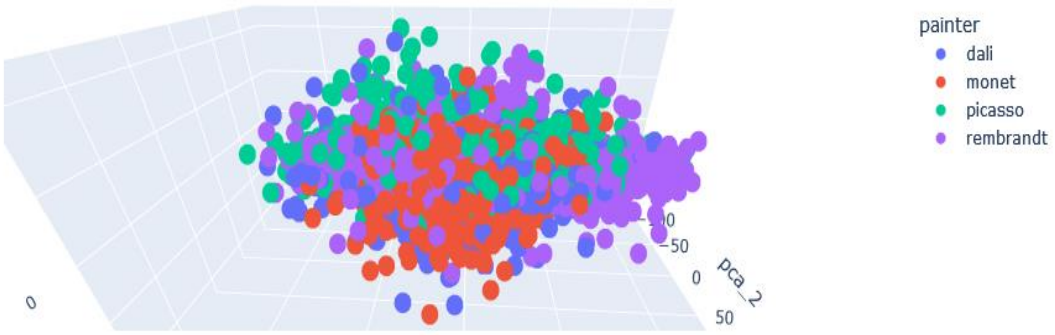

Figure 1- PCA (3d) visualization

2ν – Feature Selection

Δοκιμάσαμε να τρέξουμε αναλύσεις με λιγότερα gabor filters, χωρίς τα hog ή και με αυτά, και με άλλες διαστάσεις των εικόνων, και καταλήξαμε να παίρνουμε τα καλύτερα αποτελέσματα (δεδομένου και του χρόνου) (κατά μέσο όρο +1.5 -2 % στην ακρίβεια των προβλέψεών μας) για τις διαστάσεις 64x64 και με όλα τα υπόλοιπα features παρόντα. Λόγω περιορισμένου χρόνου δεν προλάβαμε να αφήσουμε έναν κώδικα που να παράγει έναν πίνακα με τις τιμές αυτές ώστε να είναι πιο εύκολη η σύγκριση – καταγράφαμε δηλαδή κάθε φορά την μέση ακρίβεια σε χαρτί και επαναλαμβάναμε την ανάλυση).

Παρόλα αυτά, υλοποιήσαμε με τα features που έχουμε ήδη εξάγει (pixel_values, gabor_filters, hog), εφόσον κάναμε flattened τον αρχικό πίνακα και τα μετατρέψαμε όλα σε feature vectors, έναν αλγόριθμο selectKBest, ώστε να δούμε αν χρειάζονται όλες οι στήλες μας.

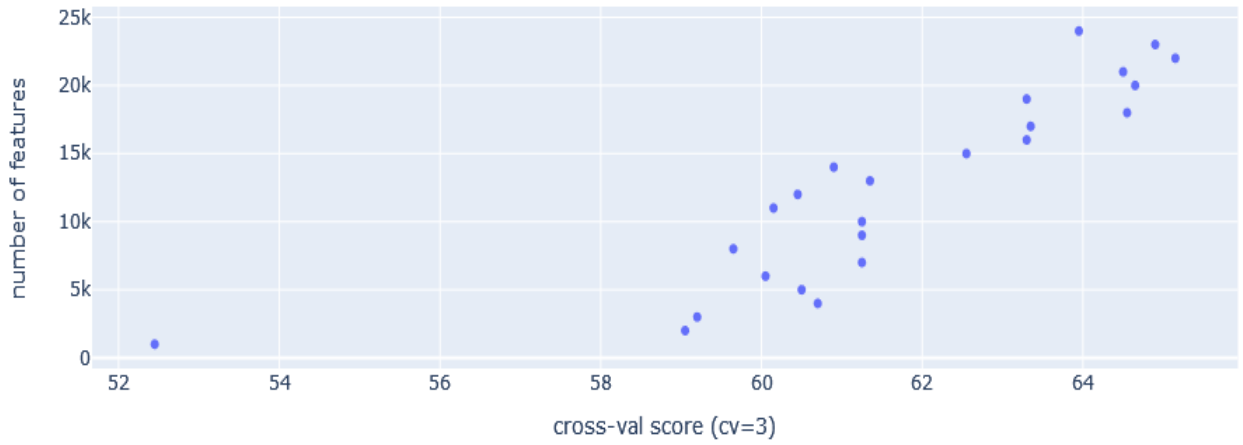

Figure 2 - SelectKBest Results

Τα αποτελέσματα δείχνουν πως για 22000 kbest features έχουμε την καλύτερη ακρίβεια. Ωστόσο, η διαφορά δεν είναι πολύ μεγάλη και στην υπόλοιπη ανάλυση τρέξαμε με όλα τα features.

3. Classifier Fitting και προβλέψεις

3α. γενική περιγραφή διαδικασίας

Συνολικά δοκιμάσαμε τρεις classifiers: Έναν Random Forest, ένα SVM και, τέλος, έναν KNN. Κάναμε ένα πρώτο fitting για να πάρουμε μία αδρή εκτίμηση της ακρίβειας και του χρόνου training καθώς και inferring, και έπειτα, χρησιμοποιήσαμε ένα Random Search CV αλγόριθμο για να κάνουμε tuning τους αλγόριθμούς μας. Ο λόγος που επιλέξαμε τον Random Search αντί για ένα αναλυτικό Search CV, είναι γιατί θεωρήσαμε την ισορροπία fine tuning/computational time του random search πιο ελκυστική για την δική μας εφαρμογή.

3β. Random Forest Classifier – Αποτελέσματα και μετρικές

Με ένα πρώτο fitting του Random Forest πήραμε τα εξής αποτελέσματα:

```
Total training time: 9.11s
Training time per example: 0.00569s
Total inference time: 0.06s
Inference time per example: 0.00015s
Test Set Accuracy : 66.5 %
```

Φυσικά η τιμές του accuracy είναι λιγάκι παραπλανητικές γιατί δεν χρησιμοποιήσαμε κάποιο folding του training dataset για το evaluation ώστε να είμαστε πιο σίγουροι για το αποτέλεσμα. Παρόλα αυτά με αφετηρία αυτήν την πρώτη εικόνα, περάσαμε στο tuning του αλγορίθμου. Χρησιμοποιήσαμε ένα πλέγμα τιμών για την εξερεύνηση όπως το εξής:

```
{'n_estimators': [100, 250, 400], 'max_features': ['sqrt'],
'max_depth': [10, 55, 100, None], 'min_samples_split': [2, 3, 5],
'min_samples_leaf': [1, 2, 4], 'bootstrap': [True, False]}
```

Και κάναμε finetune με την εντολή, χρησιμοποιώντας και kfolding για να είμαστε πιο σίγουροι για τα αποτελέσματα:

RandomizedSearchCV(estimator = rfc, scoring=None, param_distributions = random_grid, n_iter = 20, cv = 5, verbose=2, random_state=42, n_jobs = -1).

Το τελικό αποτέλεσμα για τις καλύτερες παραμέτρους ήταν το ακόλουθο:

```
{'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 4,
'max_features': 'sqrt', 'max_depth': 55, 'bootstrap': False}
```

Η ακρίβεια που επιτεύχθηκε και ο χρόνος εξαγωγής συμπεράσματος ήταν:

```
Total inference time: 0.24s
Inference time per example: 0.00061s
Test Set Accuracy : 69.25 %
```

Η διαφορά λοιπόν συνοψίζεται ως εξής:

```
Optimised model:
Total inference time: 0.06s
Inference time per example: 0.00016s
Test Set Accuracy : 69.25 %

Base model:
Total inference time: 0.07s
Inference time per example: 0.00017s
Test Set Accuracy : 66.5 %
```

Πιο αναλυτικά, για να πάρουμε μία εικόνα του πόσο καλά τα πήγε ο ταυτοποιητής μας, εξήγαμε τις εξής μετρικές/γραφικές:

Για τα train data:

	precision	recall	f1-score	support
dali	0.62	0.61	0.61	107
monet	0.70	0.74	0.72	93
picasso	0.66	0.63	0.65	101
rembrandt	0.80	0.80	0.80	99
accuracy			0.69	400
macro avg	0.69	0.70	0.69	400
weighted avg	0.69	0.69	0.69	400

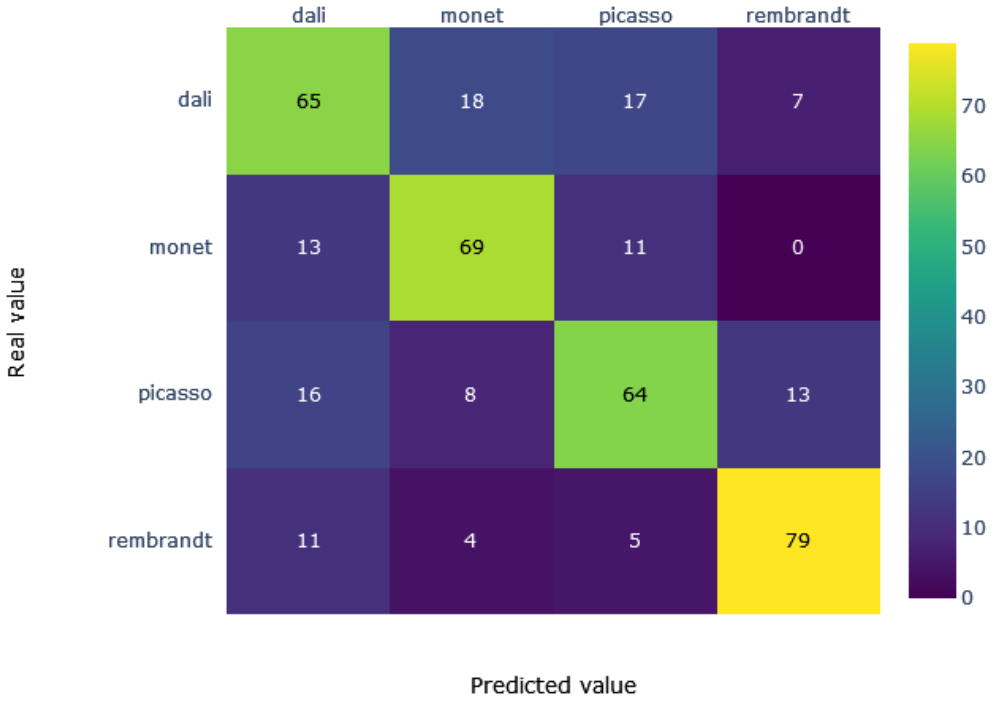

Figure 3 Random Forest Confusion Matrix

Για να εξάγουμε τις γραφικές αυτές χρησιμοποιήσαμε τις functions pr_rec_curve(), conf_m(), createRocCurve(). Όλες αυτές οι function βρίσκονται στο κομμάτι του jupyter notebook που έχει τίτλο «Metric Functions».

Figure 4 Roc Curve - Random Forest

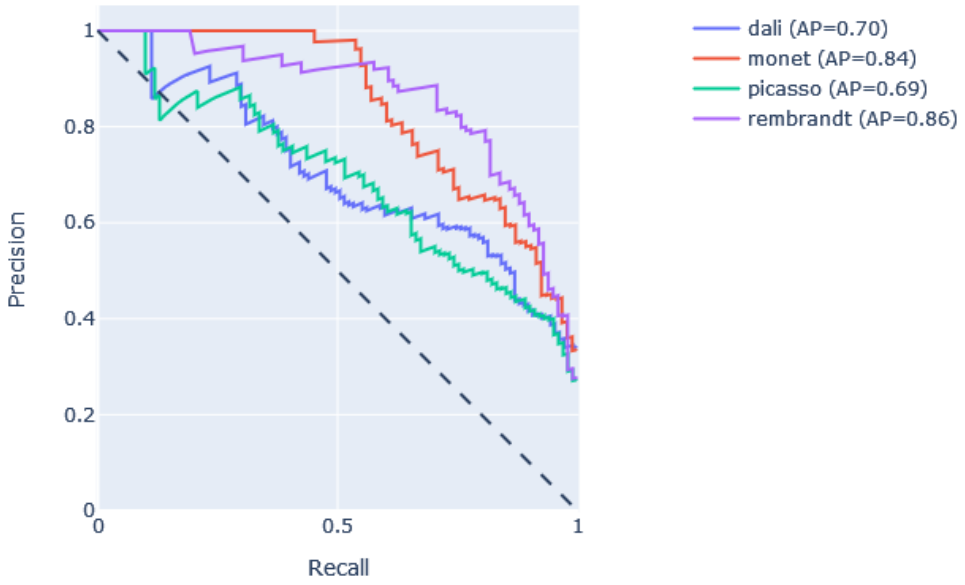

Figure 5 Precision - Recall, Random Forest

Για τα Demo Data (τα οποία λειτουργούν και ως ένα δεύτερο έμμεσο validation, αφού ποτέ δεν τα «είδε» ο random forest όταν εκπαιδευόταν):

	precision	recall	f1-score	support
dali	0.55	0.52	0.54	100
monet	0.76	0.75	0.75	100
picasso	0.59	0.63	0.61	100
rembrandt	0.81	0.81	0.81	100
accuracy			0.68	400
macro avg	0.68	0.68	0.68	400
weighted avg	0.68	0.68	0.68	400

Συμπεράσματα για τον Random Forest

Σαν πρώτη παρατήρηση βλέπουμε πως ο Random Forest Classifier, δεδομένου του όγκου των δεδομένων και της δυσκολίας του προβλήματος, τα πήγε σχετικά καλά. Βλέπουμε πως από το training/validation έβγαλε μία ακρίβεια κοντά στο 69%, και όταν τον χρησιμοποιούμε στα demo δεδομένα, που δεν τα έχει δει ποτέ, πλησιάζει την ακρίβεια που πέτυχε στο training. Από αυτό συμπεραίνουμε ότι δεν έχει κάποιο πρόβλημα overfitting ή underfitting. Συγκεκριμένα, από το confusion matrix βλέπουμε πως τα περισσότερα misclassify τα έχει για τον Dali, 42 false positive συνολικά από ένα σύνολο 107 *predicted as positive*. Αυτή η αδυναμία μεταφέρεται και όταν δοκιμάζουμε τον classifier στα demo

δεδομένα. Από την άλλη βλέπουμε το recall να είναι σχετικά όμοιο, δηλαδή, αυτά που προέβλεψε ως θετικά σε σχέση με αυτά που όντως είναι θετικά.

Δεύτερον, βλέποντας τις γραφικές για τα roc curves και τα precision recall curves (και εστιάζοντας κυρίως στο πρώτο, αφού έχουμε ένα balanced dataset), συμπεραίνουμε ότι οι πιθανότητες που έχει το μοντέλο μας να ταυτοποιήσει ένα τυχαίο – θετικό δείγμα σε σύγκριση με ένα τυχαίο – αρνητικό είναι αρκετά καλές για όλες τις κλάσεις (auc >= 0.85). Παρόλα αυτά, από το Precision Recall Curve, βλέπουμε πως το Ap είναι μέτριο για ορισμένες κλάσεις (Picasso & Dali).

Γενικά πιθανολογούμε ότι θα μπορούσαμε να δοκιμάσουμε τα εξής για να βελτιώσουμε περαιτέρω την ακρίβεια:

1) Να χρησιμοποιούσαμε και τα demo δεδομένα στο training για αυτόν τον σκοπό (θα εκπαιδεύαμε και με αυτά, κάνοντας folding επίσης).
2) Μία ακόμη εκτενέστερη έρευνα των hyper parameters.
3) Ένα καλύτερο «καθάρισμα» του dataset, ώστε να μην περιέχει ημιτελείς πίνακες από κάθε ζωγράφο, για παράδειγμα.

3ν. SVM Classifier

Με ένα πρώτο αδρό fitting για τον SVM classifier, παίρνουμε τα εξής:

```
Total training time: 33.97s
Training time per example: 0.02123s
Total inference time: 14.97s
Inference time per example: 0.03744s
Test Set Accuracy : 64.75 %
```

Κάνοντας ένα tuning με τους ακόλουθους παραμέτρους:

{'C': [1, 10, 100], 'gamma': [1, 0.1, 'scale'], 'kernel': ['rbf']}

rand_search = RandomizedSearchCV(svc, param_distributions = rand_list, n_iter = 9, cv = 3, scoring='accuracy', verbose = 10, n_jobs = 1)

παίρνουμε τα παρακάτω αποτελέσματα:

```
best parameters: {'kernel': 'rbf', 'gamma': 'scale', 'C': 10}
```

```
Total inference time: 14.39s
Inference time per example: 0.03599s
Test Set Accuracy : 65.5 %
```

Παρατηρούμε βελτίωση και στον training χρόνο και στον inference και στην ακρίβεια.

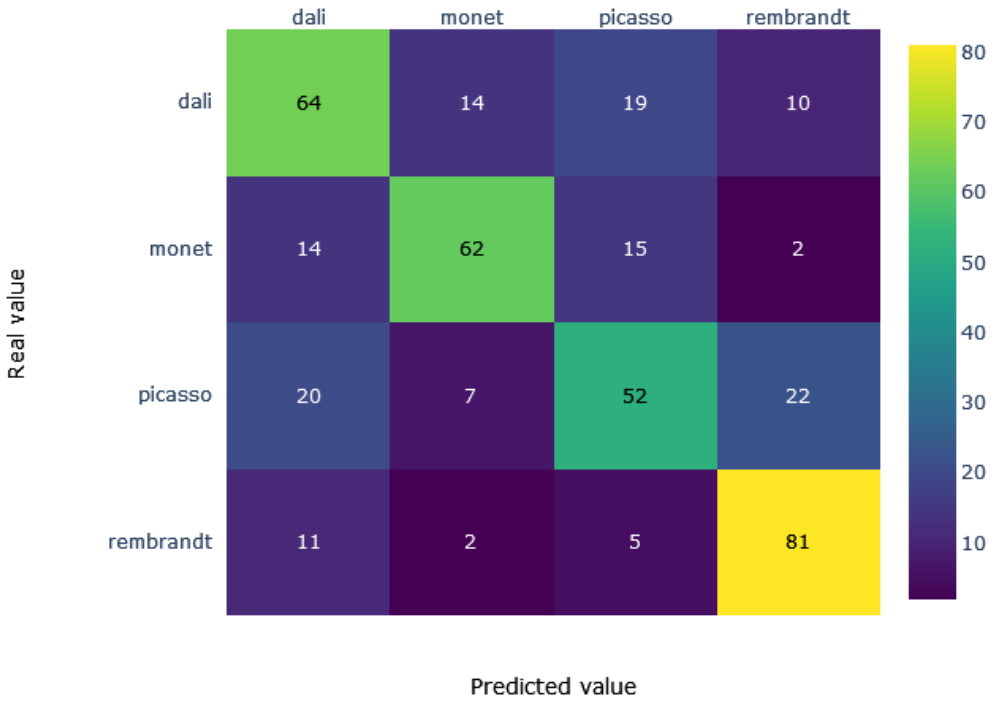

Figure 6 - CM Matric for SVM

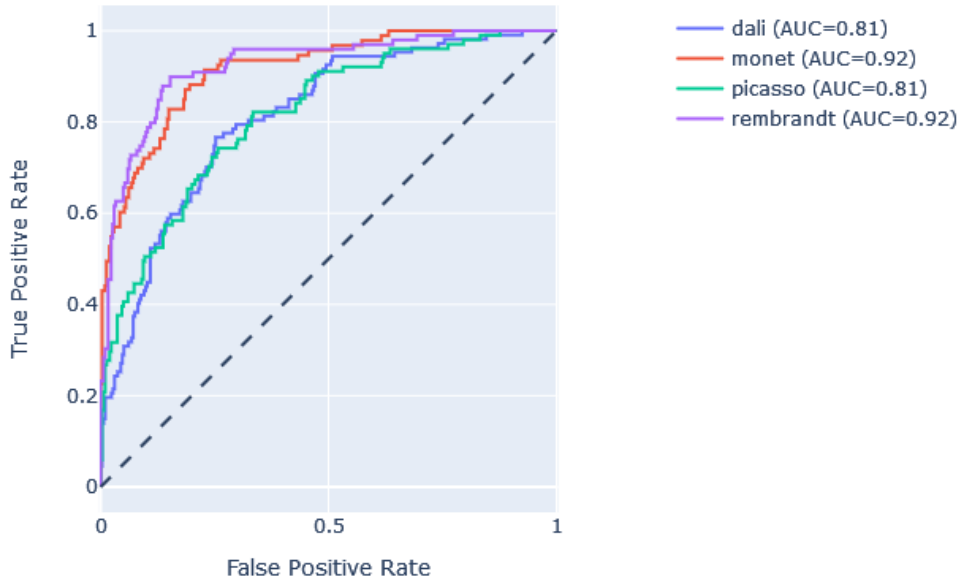

Figure 7 - ROC curves SVM

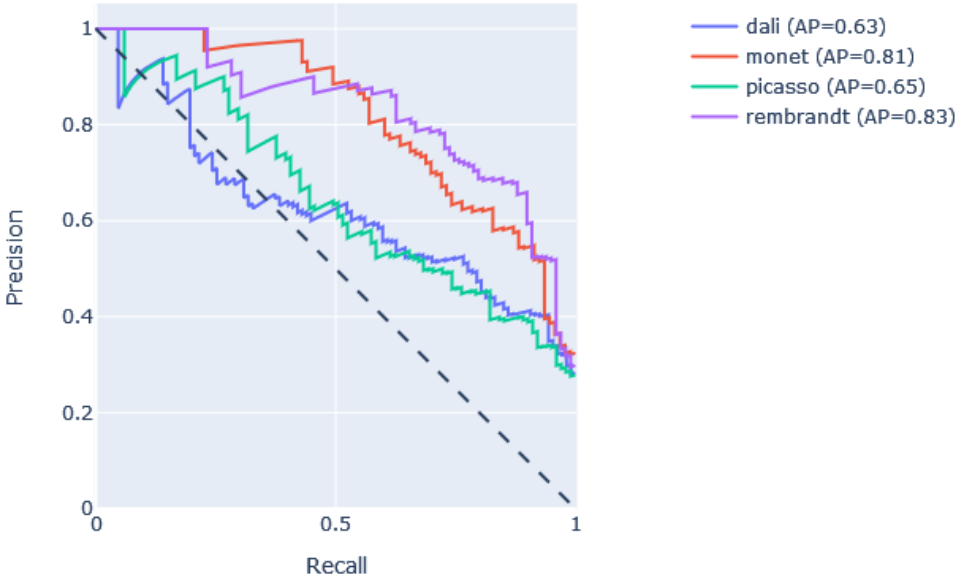

Figure 8 - Precision Recall SVM

Τέλος, στα demo δεδομένα έχουμε τις εξής μετρήσεις:

	precision	recall	f1-score	support
dali	0.53	0.51	0.52	100
monet	0.72	0.76	0.74	100
picasso	0.58	0.52	0.55	100
rembrandt	0.73	0.78	0.75	100
accuracy			0.64	400
macro avg	0.64	0.64	0.64	400
weighted avg	0.64	0.64	0.64	400

Συμπεράσματα για τον SVM

Παρατηρούμε ότι, σε κάθε μετρική, φαίνεται πως αυτός ο svm classifier είναι λιγάκι χειρότερος από τον Random Forest (ειδικά στις καμπύλες precision – recall).

3$^{\delta}$. KNN Classifier

Σαν μία αρχική εικόνα της συμπεριφοράς αυτού του classifier στο συγκεκριμένο dataset, κάναμε fitting έναν generic knn classifier του sklearn και πήραμε τα εξής πρώτα αποτελέσματα:

```
Total training time: 0.42s
Training time per example: 0.00026s
Total inference time: 0.84s
Inference time per example: 0.00211s
Test Set Accuracy : 51.5 %
```

Έχοντας κάνει tuning ξανά χρησιμοποιώντας μία μέθοδο randomized search cv, πήραμε τα αποτελέσματα:

```
Total inference time: 182.21s
Inference time per example: 0.45551s
Test Set Accuracy : 56.75 %
```

Τα οποία βελτίωσαν μεν την ακρίβεια στις προβλέψεις, αλλά με πολύ μεγάλο τίμημα στον χρόνο που χρειάζεται για την εξαγωγή ενός συμπεράσματος. Παρόλα αυτά, ούτε η ακρίβεια στις προβλέψεις εντυπωσιάζει, συνεπώς αυτός ο classifier φαίνεται να μην «ταιριάζει» στο dataset μας. Συνεχίζουμε παραθέτοντας τις γραφικές παραστάσεις:

Για τα train δεδομένα:

	precision	recall	f1-score	support
dali	0.60	0.46	0.52	107
monet	0.52	0.83	0.64	93
picasso	0.82	0.14	0.24	101
rembrandt	0.57	0.88	0.69	99
accuracy			0.57	400
macro avg	0.63	0.58	0.52	400
weighted avg	0.63	0.57	0.52	400

Για τα demo δεδομένα:

precision	recall	f1-score	support
dali	0.46	0.36	0.40	100
monet	0.45	0.76	0.56	100
picasso	0.64	0.07	0.13	100
rembrandt	0.53	0.75	0.62	100
accuracy			0.48	400
macro avg	0.52	0.49	0.43	400
weighted avg	0.52	0.48	0.43	400

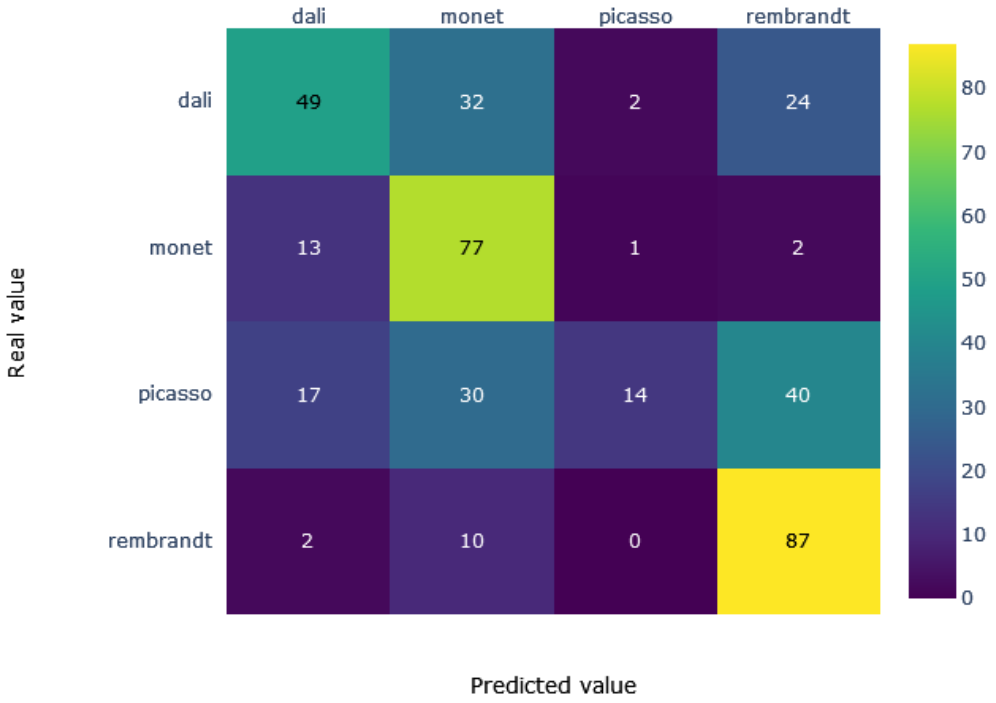

Figure 9 - KNN confusion matrix

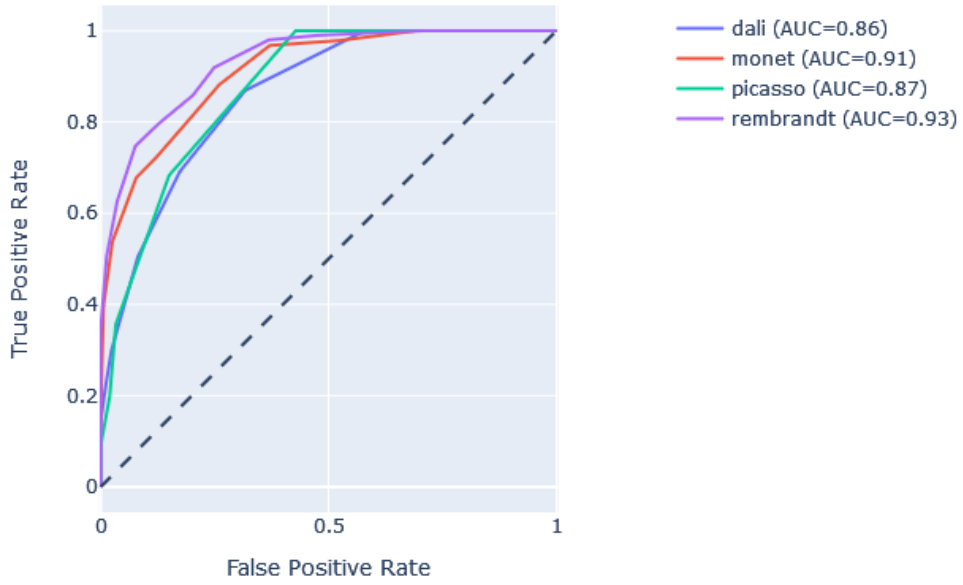

Figure 10 - ROC curves, Knn

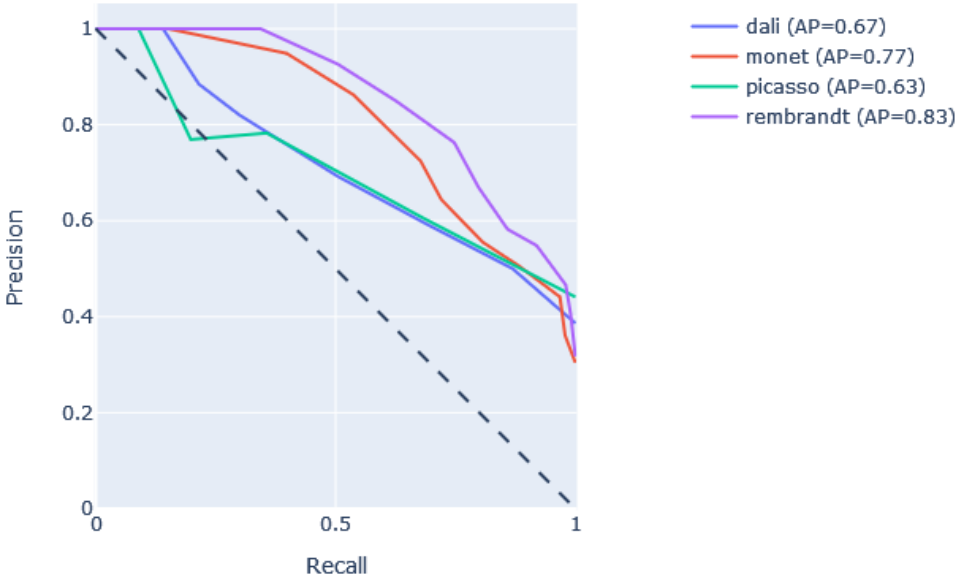

Figure 11 - Precision Recall KNN

Συμπεράσματα για τον KNN

Αυτός ο κατηγοριοποιητής δεν μας άφησε ιδιαίτερα ικανοποιημένους. Παρατηρούμε ότι στα demo δεδομένα το accuracy έχει ιδιαίτερη πτώση. Επίσης, οι κακές επιδόσεις φαίνονται και στις γραφικές, όπως το confusion matrix.

Τελικές Παρατηρήσεις κεφαλαίου

- Από όλους του classifiers που δοκιμάσαμε, αυτός που φάνηκε να λειτουργεί καλύτερα ήταν ο Random Forest. Η διαφορά ήταν αισθητή όχι μόνο στις μετρικές της ακρίβειας και στις ανάλογες γραφικές, αλλά και στον χρόνο που απαιτεί υπολογιστικά να εκπαιδευτεί και να εξάγει κάποιο συμπέρασμα (training time – inference time).
- Πιθανότερη περισσότερη ακρίβεια σε όλους του classifiers θα μπορούσε να επιτευχθεί με μία πιο εξαντλητική εξερεύνηση για optimal παραμέτρους. Αυτό θα μπορούσε να γίνει, για παράδειγμα στο SKLEARN, με ένα GridSearchCV, το οποίο θα απαιτούσε όμως πολύ περισσότερο χρόνο. Δεν το επιλέξαμε γιατί εστιάσαμε περισσότερο στην εξοικείωση με τα εργαλεία παρά στην εξαντλητική έρευνα για τον καλύτερο classifier.
- Επίσης, κάτι που ενδεχομένως επηρέασε τα τελικά αποτελέσματα ήταν το dataset το οποίο δεν ήταν απόλυτα τακτοποιημένο. Δηλαδή, στις εικόνες από τους πίνακες, έτσι όπως τις φέραμε, πολλές φορές υπήρχαν και κάποια προσχέδια ή ημιτελείς εκτελέσεις των πινάκων, από τους ίδιους τους καλλιτέχνες βέβαια. Ένα προσεχτικό ψάξιμο και μία τακτοποίηση του dataset, όχι υπολογιστικά, πράγμα που προσέξαμε να είναι πολύ καθαρό, αλλά όσο αφορά τα raw δεδομένα, θα βελτίωνε τα αποτελέσματα.
- Τέλος, η αύξηση της ανάλυσης των εικόνων, και η εισαγωγή περαιτέρω features (sobel, surf, sift, orb descriptors), σίγουρα θα βελτίωνε την ακρίβεια των προβλέψεων (όπως φυσικά θα αύξανε και τον υπολογιστικό χρόνο που θα απαιτούνταν για την εκπαίδευση των μοντέλων μας).

4. Demo και παρουσίαση

4^{α}. demo δεδομένα και ένα classification

Όσο αφορά τον κώδικα demo, επιλέξαμε από την αρχή της εργασίας περίπου εκατό πίνακες από κάθε έναν από τους τέσσερις ζωγράφους, και τους ξεχωρίσαμε ώστε να μην τους δουν ποτέ οι classifiers μας κατά την εκπαίδευση – τεστάρισμά τους. Στο τέλος του jupyter notebook “feature_extraction” έχουμε τον σχετικό κώδικα όπου μπορούμε να κάνουμε load οποιονδήποτε από τους τρεις classifiers που εκπαιδεύσαμε προηγουμένως, και να τον δοκιμάσουμε σε αυτά τα δεδομένα. Ήδη από τα προηγούμενα ερωτήματα έχουμε παραθέσει κάποιες μετρικές για το πως τα πήγαν οι classifiers μας σε αυτά. Επιπρόσθετα, έχουμε υλοποιήσει με κώδικα στο μέρος του dataset με τίτλο “classifier URL tool” ένα εργαλείο όπου μπορεί ο χρήστης να βάλει ένα url που να παραπέμπει σε μία εικόνα και να πάρει μία πρόβλεψη για τον καλλιτέχνη που τον σχεδίασε.

4^{β}. demo classifier tool

Έχουμε υλοποιήσει και σε plotly - DASH ένα demo που δείχνει τις βασικές μετρικές για ένα dataset πινάκων στο οποίο δεν έχει εκπαιδευτεί κανένας από τους classifiers μας, και επίσης υπάρχει implementation ενός απλού εργαλείου που δέχεται το url ενός πίνακα και επιστρέφει το class predictιon. Τα αρχεία αυτά είναι τα dash_paint_02_demo.py και dash_paint02_functions.py.