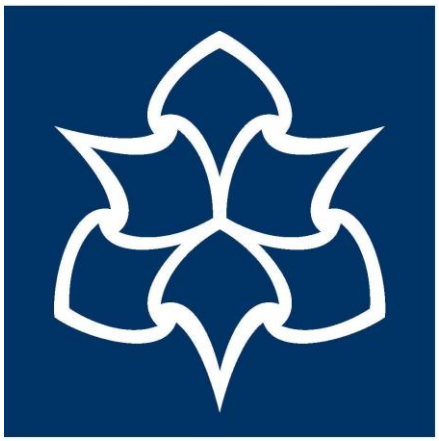
Image Texture Analysis and Classification

A REPORT SUBMITTED TO MANCHESTER METROPOLITAN UNIVERSITY FOR
THE DEGREE OF BACHELOR OF SCIENCE
IN THE FACULTY OF SCIENCE AND ENGINEERING



2021

By
John Carter
School of Computing,
Mathematics and
Digital Technology

Declaration

No part of this project has been submitted in support of an application for any other degree or qualification at this or any other institute of learning. Apart from those parts of the project containing citations to the work of others, this project is my own unaided work.

Signed: John Ward Carter

Date: 26/07/2021

Acknowledgements

Safwán Anjum-James

Harry Carter

Debra Core

Dr Nicholas Costen

Fraser Connor

Jacob Creed

Irfan Hanafi

Rory Hillyer

Cristina Morales-Bages

Hayley McDermott

Arman Najafian

Emma Preston

Christopher Riley

Abbreviations

ITA	-	Image Texture Analysis
ML	-	Machine Learning
SIFT	-	Scale Invariant Feature Transform
GTVE	-	Geometric Total Variation Energy
TVG	-	Total Variation Geometry
KNN	-	K-Nearest Neighbours
LDA	-	Linear Discriminant Analysis
CH	-	Colour Histogram
SVM	-	Support Vector Machine
SVT	-	Support Vector Tracking
CNN	-	Convolutional Neural Network
DCNN	-	Deep Convolutional Neural Network
ReLU	-	Rectified Linear Units
VGG	-	Visual Geometry Group
FL	-	Fully Connected Layer

EWM	-	Element-Wise Multiplication
EWS	-	Element-Wise Sum
BP	-	Bilinear Pooling
CBP	-	Compact Bilinear Pooling
CMP	-	Compact Multilinear Pooling
FASTLAB	-	Features from Accelerated Segment Test
DBSCAN	-	Density-Based Spatial Clustering of Applications with Noise
ANOVA	-	Analysis of Variance

List of Tables

List of Figures

Chapter 1 - Abstract

Image texture analysis is the process of extracting data from an image in order to classify it, based on patterns that can be found such as the surface properties like type of material it is, or how that material behaves when lit, in other words the colour or reflectiveness. This allows for the object to be classified. The goal of this work is to analyse the relationship between image filtering techniques, Neural Networks, and Classifiers. This project will be using machine learning and texture analysis to identify different fragments of pottery using several different methods and comparing the accuracy and performance, with a view of speeding up the manual process of categorising these pieces.

Chapter 2 - Literature Review

2.1 Definition of Problem

Image texture analysis (ITA) is difficult because computers like uniformity and real objects are not uniform

The difficulty will be in training a model that will be able to differentiate things like the type of clay and any imperfections it might have, the markings if it has any, and the size and shape of the object.

In order to do this the features of the image must be broken down into data points based on the properties specific to our samples, in this case pottery sherds (this is the term for shards that come from ceramics, and will be used going forward). This will involve identifying what features (such as tactile features like the material composition, or more visual features like colour and markings) will allow the computer to make predictions and place each image into a class which again will need to be identified from the sample images or from manual examination beforehand.

All parts of the product (image filtering, neural networks, and classifiers) will need to be tested and tuned to identify the best combinations to ultimately achieve the most accurate and performant methods of classification/analysis.

2.2 Frameworks, Libraries, and Networks

2.2.1 OpenCV

OpenCV is an open source cross-platform library that specialises in computer vision, image processing and analysis and contains many computer vision algorithms (*OpenCV: Introduction*, n.d.).

It is designed with a modular structure and some of these are relevant to the work to be undertaken, one of which is the image processing module (imgproc) which is used for image

filtering to prepare data for classification using tools such as geometric image transformations and colour histograms.
Another is the 2D features framework (features2d) which can be used to obtain features and descriptors.
The last of interest is the object detection module (objdetect) which can then analyse the prepared data and detect the needed object itself from that data. It can also predict what predefined class the object most likely falls into.

2.2.2 ImageAI

ImageAI (Olafenwa, 2021) is an open source Python library that uses libraries like Tensorflow, Keras, and OpenCV to easily run computer vision projects with minimal code. This is useful for rapid prototyping.
It uses 4 algorithms for image prediction, MobileNetV2, ResNet50, InceptionV3, and DenseNet121. These are all trained using the ImageNet-1000 Dataset.

2.2.3 Tensorflow

Tensorflow is an open source machine learning platform developed by Google that allows users to access Google's Deep neural networks (*tensorflow/tensorflow*, 2021). There are several models for image classification and object detection, and for the purposes of computer vision uses ResNet and the fork ResNet-RS, both deep residual learning models for image classification.

2.2.4 AlexNet

AlexNet is a deep CNN (DCNN) designed to improve on standard CNN design in a few ways, one by taking advantage of two linked GPUs by splitting the load evenly across them. It also modified the standard practice of pooling neighbouring data by introducing overlapping. Finally, it uses Rectified Linear Units (ReLU) instead of the standard tanh function. All these combined led to drastically reduced training times (Krizhevsky et al., 2017:3-5).

2.2.5 Residual Network

The Residual Network (ResNet) is a DCNN that introduces far more layers than AlexNet, which had 8 neural network layers, 5 convolutional and 3 fully connected. ResNet can have layer counts ranging from 20 to 1202(He et al., 2016). While normally this might make the error rates rise due to overfitting the data, ResNet uses something called a Residual Block, which allows the network to be much deeper than previous CNNs without the increased risk of overfitting. There have been several variations of ResNet, with one of the more popular ones being ResNet50 which uses 50 layers.

2.2.6 Visual Geometry Group

The Visual Geometry Group (VGG) is a DCNN that uses AlexNet as its base. It differs from AlexNet by adding more ReLU units with fewer parameters. It also uses small size convolutional layers and more weight layers. This allows it to perform better than AlexNet and is one of the most widely used CNNs.

2.3 Classification Models and Descriptors

There are several methods that have been commonly used to identify and classify pottery sherds, with advantages and disadvantages to each type which will be expanded upon.

All models must go through the same phases, the data must be obtained, then prepared for analysis, then the model must be trained on a subset of the data, and then the model will predict/classify the data on the remaining data, sometimes referred to as the holdout.

The classification descriptors and models discussed here will be limited to the ones used in the other papers pertaining to machine learning and pottery sherds.

2.3.1 Descriptors

2.3.1.1 Scale-Invariant Feature Transform

Scale-Invariant Feature Transform (SIFT), is a widely used method used for image-feature generation by ‘transforming an image into a large collection of local feature vectors’. (Lowe, 1999: 1). The model is unaffected by changes to scale, rotation, and transformation and such, making it a rather robust image feature generator.

2.3.1.2 Geometric Total Variation Energy

Geometric Total Variation Energy (GTVE) is a texture descriptor based on the principle that after creating image derivatives and gradients, these can be used to compare to the original source, and any areas with high variation from the original are considered points of interest and can be used to create features for later analysis. The Total Variation Energy (TVG) descriptors are similar to SIFT. Subdivision is used on the image to calculate the TVG energies, and each subdivided window is concatenated into a 32-dimensional feature vector, in contrast to SIFTs 128 (Smith et al., 2010:51).

2.3.1.3 Colour Histograms

A Colour Histogram is a digital representation of colour in images. To start, each pixel is given a colour value, one for red, green, and blue with the higher value meaning the more of that colour is in that pixel. This gives us a pixel datapoint with three features.
The histogram ‘denotes the joint probabilities of each intensities of the three colour channels’ (Smith and Chang, 1996: 4). This can then be transformed into a single histogram.
Because the only features are pixel colour values, it ignores any textural information, and as such can only be considered part of a solution to the detection and classification of sherds.

2.3.2 Models

2.3.2.1 K-Nearest Neighbours

The K-Nearest Neighbours (K-NN) model is a simple model that aims to categorise data by first plotting the data to be compared, and then predicting each datapoint by comparing it to the closest data points that surround it, its nearest neighbours. The distance between objects can be calculated in different ways, for example Euclidean distance which is best for data with numerical values, or Manhattan distance which is best for measuring distance between two vectors.

It is a rather simple algorithm as the only variable that can be tuned is K, how many neighbours are considered for each data point.

It is suited to sherd classification because of its O(1) (constant) short training time, however the testing time is O(n) (linear) which might make it unsuitable for larger testing datasets as the time taken rises with the number of items to process.

2.3.2.2 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) takes a similar plot of data as KNN, but instead of calculating distance between points, LDA attempts to reduce the dimensionality of the data by creating an axis across the data. All data is now plotted along this axis, the algorithm attempts to maximize the distance between the mean value of each category, while also minimising the variation of each category (called “scatter”).

The number of components for this dimensionality reduction can be set, but must always be at most n-1, where n is the number of classes. This makes it ideally suited to multi-class datasets.

2.3.2.3 Support Vector Machine/Tracking

The Support Vector Machine (SVM) is a classifier that attempts to categorise data by creating a hyperplane that separates the categories, drawn in between two data points of differing classes that are the closest together on a plot. These closest data points are the “Support

Vectors" and the further away a datapoint is, the higher the confidence value that it belongs to the label on that side of the hyperplane. Because of this, it works best with smaller datasets.

2.4 Related work

2.4.1 Classification of Archaeological Ceramic Fragments Using Texture and Color Descriptors - Smith et al.

This study opted to determine the effects of differing feature descriptors: the pre-existing SIFT and a new implementation of GTVE first described by Burchard(Burchard, n.d.), both with and without the use of a colour histogram to identify colour as well as texture.
The sherds used were of 18th and 19th century American make and were photographed from a top down view of the outside of the sherd.
The images were then prepared using colour histograms, SIFT and GTVE, then a K-NN model was applied to the data, testing for differences in the two models intrinsically, and with colour histograms combined for a larger feature set.
When the model was applied to the data that did not have colour histograms in the features, SIFT had an accuracy of 63% and GTVE only had 49%. When the data had colour features included, that accuracy rose by 25% and 53% respectively (Smith et al., 2010:54). The paper concludes that GTVE shows similar accuracy to SIFT, but reduces the dimensionality from 128 to 32, reducing the needed information and speeding up the process.

2.4.2 Gabor Filter-Based Texture Features to Archaeological Ceramic Materials Characterization - Abadi et al.

This study revolves around testing the effect that using a Gabor filter has on image analysis and classification using Linear Discriminant Analysis (LDA) and K-Nearest Neighbours (K-NN) as models.
The sherds were of Egyptian origin from the Abu Roach site, with some coming from the New Kingdom Era, the start of the Ptolemaic Era, and the beginning of the Arabic Period (Abadi et al., 2012:335). 599 Photos were taken along the break (with four sub images each for 2396

samples), and the goal was to identify the sherds from the surface texture (there were several types, ceramic paste/fabric, and ceramic surfaces, with several subcategories) along the break, and whether there was paste or fabric on the sherd.

For feature extraction, a Gabor filter-based approach was used (Mehrotra et al., 1992), which involves analysing the frequency and orientation of each sub-image by detecting any edges in the direction set by the user, as well as the scale. For each response image, the first three moments were captured as features to use.

With prepared data, the K-NN and LDA classifiers were used, with K-NN providing overall best results, which the paper asserts may be due to LDA not performing as well with diffuse surfaces (Abadi et al., 2012). The paper posits that the results could be improved by using the classifier that scored the highest accuracy for each of the types of fragment.

2.4.3 Classification of Engraved Pottery Sherds Mixing Deep-Learning Features by Compact Bilinear Pooling

This paper sets out to examine the relationship between pre-trained neural networks(AlexNet, ResNet50, VGG16, and VGG19), classifiers (LDA, Support Vector Tracking(SVT)(Avidan, 2004), and Fully Connected Layer (FL)) and pooling strategies (Concatenation(Concat), Element-Wise Multiplication(EWM), Element-Wise Sum(EWS), EWM+EWS, Bilinear Pooling(BP), and Compact Bilinear Pooling(CBP)).

The sherds to be analysed are rough sherds from Saran, France from the Middle Ages, and are engraved with repeating geometric patterns. 888 digital patterns were used for sampling.

In order to prepare the data for analysis 3d scans taken by nextEngine were turned into 2d shaded views using MeshLab, then Features from Accelerated Segment Test(FASTLAB) is used on the resultant grayscale variance image and finally Density-Based Spatial Clustering of Applications with Noise (DBSCAN) grouping method detects ‘clusters with highest textured point density’ (Chetouani et al., 2020:2).

Pre-existing and pre-trained neural networks (named above) were used for feature extraction from the prepared data, and evaluated when run individually on the data, and when combined. The results were that ResNet set to 50 was the best network, and when combined using CBP for pairwise combinations and Compact Multilinear Pooling (CMP) for triplet combinations.

When these pooling methods were used accuracy increased by ~13% over their previous paper regarding Wavelet Transformation Methods (Eslami et al., 2021), and ResNet50 + VGG19 in pairwise and when used in a triplet was the most accurate overall.

Analysis of Variance (ANOVA) was then used to evaluate pooling strategies, using the most accurate pairwise combination from earlier, finding that CBP performed best, and a pairwise combination of pooling strategies was used, and CBP was retained as the most accurate solution.

Finally, the three classifiers were evaluated with all pooling strategies, and found that any combination of classifier and pooling method would yield a ~2% increase in performance (Chetouani et al., 2020:6). The most accurate combination was VGG19+ResNet combined with an FC classifier, with concatenation + SVM then CBP + LDA, and note that CBP has around 40% less parameters while still achieving a similar classification rate. They conclude that a larger dataset and more features could lead to a better classification method.

Chapter 3 - Design

3.1 Prototype

For the analysis and classification system, a basic prototype will be created to determine the types of neural networks and classifiers that will be best suited to identifying sherds. This will be a simple two-class problem, with the entire sherd dataset as the positive class, and images of things that are not pottery for the negative. Because there is only a binary choice for classification, all NNs and classifiers are expected to be highly accurate, so the goal of the prototype will be more based around testing the interactions of a neural network for feature extraction and a classifier.

3.2 Product

Based on the testing done at the prototype phase a selection of neural networks and classifiers have been selected to analyse the sherd images. In order to test the importance of colour, the images will be split into two identical sets, where one is untouched and the other has a Gabor filter applied, which will create a grayscale composite image created from several passes using different wavelengths and orientation combinations. The convolutional neural nets used will all be pre-trained because there are only 129 samples, which will not be enough to effectively train the NNs. To create the Gabor-filtered images, a separate script will be run to undertake this transformation. Once prepared the two datasets will be run independently through 3 pre-trained neural nets to generate features (the nets can predict outcomes themselves but are not in the scope of this project). The twelve feature sets will then undergo classification on two classifiers, and the resultant 12 predictions will be evaluated for accuracy and performance.

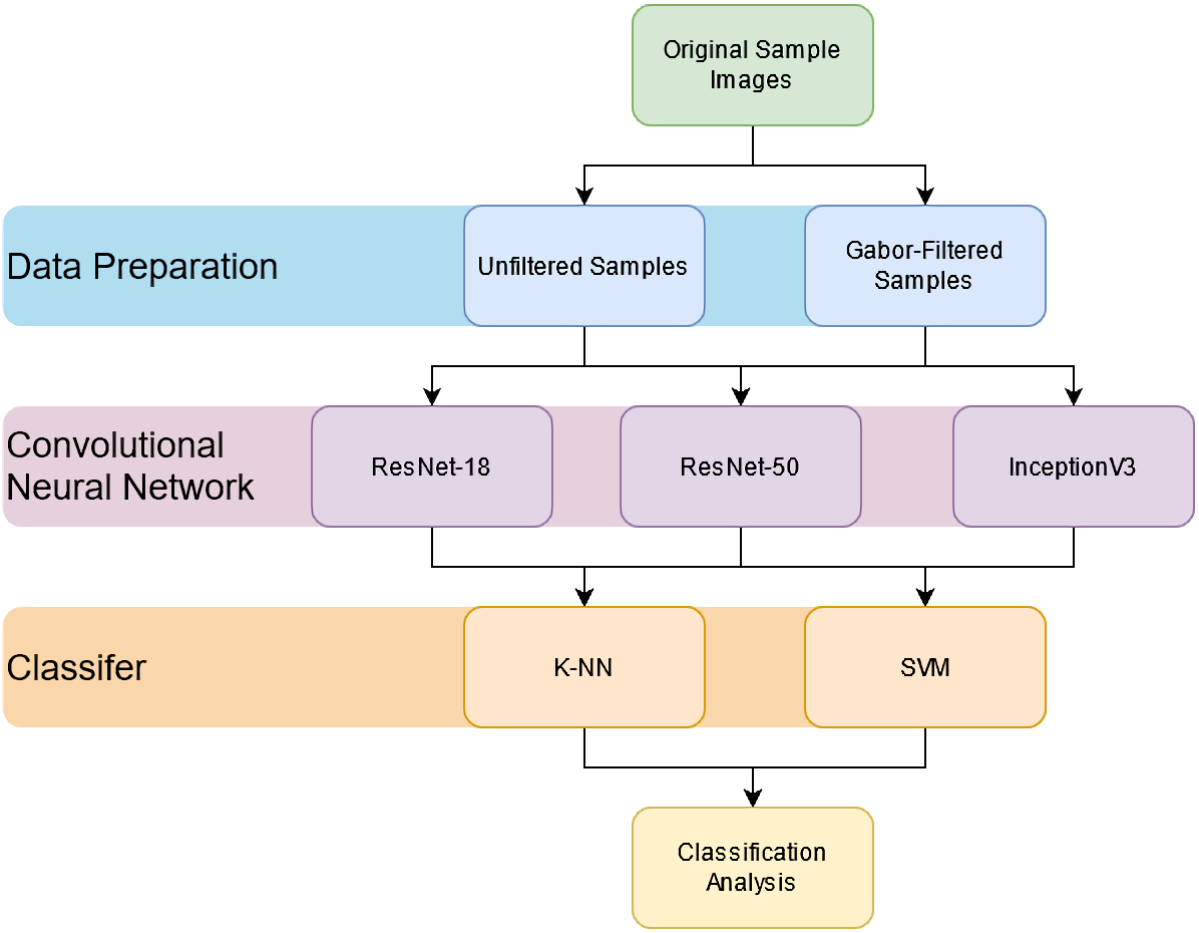


Figure 1: Planned data flow of product

Chapter 4 - Implementation

4.1 Prototype

The Prototype product, like the main product will be built in MATLAB due to its integrated support for neural networks and Gabor transforms, and will accept a folder of images, divided into subfolders based on a binary pottery fragment as the positive case, and for the negative case an equal amount of non-pottery images taken from the STL-10 Image Recognition Database (*STL-10 Image Recognition Dataset*, n.d.), taken at random from the test images. For feature extraction three neural nets were tested - ResNet-18, ResNet-50, and InceptionV3, selected because of their varying depth in terms of layers, and the two ResNets to examine the impact of layers in the feature extraction and classification. For classifiers a multiclass implementation of SVM and K-NN were tested to see if a difference could be discerned. After testing all variants separately, a difference could not be discerned between them, which is expected of such a simple problem. This does however show that the selected models will work together and will be fit to run on the main product.

Note that the methods for data preparation/ feature extraction and classifier are all derived from their being used in papers covered in the literature study. This is because those methods have already shown they are capable of analysing and categorising pottery sherds.

4.2 Product

The neural networks selected from the prototype will all be pretrained, this is because the small size of the sample images (only 129 across four classes) would make classification a more difficult task than it needs to be, and with the networks being trained on over a million images from the ImageNet database they are more robust feature extractors.

When Starting the extraction/classifying process, first the versions of the sample images due to undergo Gabor filtering must be created.

Firstly, the script will iterate over the images and apply the transformation. because the Gabor filter specifically looks for edges along a specified orientation, and to create a useful image, the image is rotated as many times as necessary. The other variable to consider is the wavelength, which will alter the width of the filter. After testing different values for these two operators, wavelengths of 2, 4, and 6 and orientations of 0, 45, 90, and 135 yielded the best results. After all these sub images have been created, the script averages the values and superimposes them together to create our final image. The resulting images will be grayscale, and as such will only have 1 colour channel. These images will not work as is with the neural nets employed because they expect an image with full RGB colour space, so after creating the composite image the script has to spoof the 3 colour channels by concatenating the same image 3 times, creating the necessary channels but without altering the filtered image, then they are written to an appropriate folder. Note that the filtered images are pre calculated before the main script is ran because the operation is quite time intensive and creating new images each time the feature generation/classification is run is not necessary as the filtered images are the same.

Then the user will move to the main project script. The script firstly creates a datastore for the untouched and filtered images and splits the samples 70/30 into training and testing stores respectively, providing enough training data to accurately classify the test portion of the images. Next the features are extracted by the CNN, these are run sequentially and separated into sections primarily for readability, and the ability to test misbehaving chunks of code. The script initialises the network and creates new datastores, this is done for two reasons:

Figure 2: Selection of samples with & without Gabor filter

Firstly, to resize the images to each network's preferred size (for example ResNet prefers a 224x224 image), and secondly so the feature set can be appropriately named for used in

evaluation later. The script will also write the generated features to file in case it is necessary to investigate the generated features. After each network has generated its features, the classifiers are run, again sequentially and in sections to aid readability.

Firstly the classifier is loaded in with the training data from feature generation and the labels, to allow the model access to the raw data, and what label those features correspond to along with any arguments to the model (for example the K-NN model allows for the number of neighbours to be changed, and testing of K=1:10 revealed 1 to be the most accurate). This allows the model to then be given the test data using the knowledge of the training data, and to form predictions about which labels each image belongs to. The final step in the classifier sections is to work out the accuracy of each network/classifier combination and display them to the user. Finally, the script will create confusion charts and write them to an appropriate folder in the project for later analysis.

Chapter 5 - Evaluation

5.1 Results Analysis

To analyse the relationship between image filtering techniques, Neural Nets, and classifiers 129 sherds total have been analysed. These fragments (a set of untouched images and a set of filtered images) were split into four labels: BB (53 images), CM (3), gw (59), and Sa (14). All pre-trained neural nets had the final five layers before the prediction layers tested and from this the following layers yielded best results: RN18 - pool5, RN50 – fc1000, IV3 – avg_pool. Then a K of 1 was set for the K-NN classifier, which yielded the best results when prototyping. To ensure that the data is not the result of a fluke or random chance skewing the odds, each of the twelve predictive model combinations were ran four times and averaged. With that in mind, across all variations and combinations, the total average accuracy of correctly predicting a sherd is 57.52%.

All Run Average	Unfiltered Images	Gabor Filtered Images	Individual Network/ Classifier Average	Total Classifier Average
ResNet-18 + KNN	0.5231	0.5128	0.5179	54.02%
ResNet-50 + KNN	0.5128	0.5333	0.5231	
InceptionV3 + KNN	0.6154	0.5436	0.5795	
ResNet-18 + SVM	0.6564	0.5692	0.6128	61.03%
ResNet-50 + SVM	0.6462	0.5692	0.6077	
InceptionV3 + SVM	0.6359	0.5846	0.6103	
Data Preparation Average	59.83%	55.21%	57.52%	58%

Table 1: Average accuracies across all runs - n.b. orange is the total average accuracy

Table 1 teaches us that One; on average filtered images will be classified ~5% less accurately regardless of which combination is used, the filtered images were being correctly predicted at a relatively similar rate, but never equals or beats the datasets using the untouched colour images, suggesting that colour is a very important feature for the computer vision pertaining to pottery classification. Two: that any algorithm combination involving the SVM classifier was ~7% more accurate which proved to be a common denominator across all runs.
Thirdly, table 2 shows in more detail the accuracies of each NN across all its different model parings and filter pairings.

What's important to note here is that both ResNets have an identical average accuracy, indicating that increasing the layer count and thereby the complexity of the neural net does not necessarily improve the accuracy of the model. The most accurate NN would appear to be InceptionV3, with a ~4% increase in accuracy for any classifier/data/extraction combination that included it.

All Run Network Average	Unfiltered Images	Gabor Filtered Images	Average Network Accuracy
ResNet-18	0.7372	0.6763	70.67%
ResNet-50	0.7244	0.6891	70.67%
InceptionV3	0.7821	0.7051	74.36%
Data Preparation Average	74.79%	69.02%	72%

Table 2: comparing average accuracies by neural net

Next we will look at the breakdown of the product's accuracy by class label, note that all confusion charts/data used here for the breakdown by class are from the final set of runs only and not multiple run averages to smooth out the data, so it is possible that some of this data is an outlier and not representative of the predictions as a whole.

Total Predictions					
True Class	**BB**	133	1	2	56
	CM	4	8		6
	Sa	6		30	12
	gw	32	4	8	172
		BB	**CM**	**Sa**	**gw**
		Predicted class			

Table 3: Average class accuracy across all models

Total Average Accuracy by Class	
Class	**Accuracy %**
BB	**62.74%**
CM	**16.67%**
Sa	**53.57%**
gw	**72.88%**

Table 4: Accuracy across all 12 model combinations

From these results we can see that the most accurately predicted class is gw at ~73%, which is to be expected as it is the largest dataset at 59 images, indeed the accuracy does strongly correlate with number of samples, with the least correctly labelled subset (by a large margin) being CM at only 3 images and an accuracy average of just 16.67%, but recall that the data is split 70/30, meaning two images are in the training set and only one image is in the test data, meaning there just isn't enough data for the models to go on, even if they have been pre-trained. Overall, the data shows that sample size may be the defining limitation of the dataset.

Total Average					
True Class	BB	11.08	1.00	1.00	5.09
	CM	1.00	1.00		1.00
	Sa	1.00		2.50	1.50
	gw	2.67	1.33	1.14	14.33
		BB	CM	Sa	gw
		Predicted class			

Table 5: confusion matrix average (all models)

One final thing to note is that the highest frequency of incorrect answers was when a gw fragment is classed as a BB, and vice versa. While it is possible that this is just a statistical anomaly from the fourth run (where all class label data has been generated from), it is more likely that because they are the largest data sets, at 53 and 59 respectively there are just more predictions being made from this class due to having more training data.

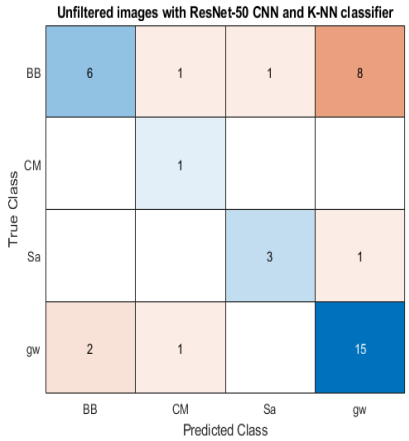


Figure 3:A confusion chart representative of the results

Based on the data collected and evaluated, this paper must conclude that the best combination of methods for the purpose of identifying sherds is using unfiltered images, the InceptionV3 convolutional neural network, and the Support Vector Machine classifier.

5.2 Reflection

After evaluation of the data generated for this product, it is only right to evaluate the sample set provided, the methodology, and the process/final product as there are some conclusions to be drawn from them.

Firstly, the classification confidence (in other words the accuracy) across all combinations ranges from ~85% to just ~54%, which is at best a rather robust classification system, but with some results hovering close to sub 50 %, a less than desirable spread. This is not believed to be a limitation of the networks/classifiers themselves; it is however indicative that the settings selected as hyperparameters may not be tuned correctly for this task. It should be noted that although accuracy could be increased (at the very least the minimum accuracy average could be brought up closer to the maximum) this would run the danger of over/underfitting the data by chasing after a "perfect" classification system, and significantly increase the scope of the problem without having a significant benefit.

Of course not averaging out the results of the runs on a class-by-class basis may have resulted in some data not being representative of the overall model, but as the data was not captured at runtime this is something that cannot be fixed, but is a mistake that would be improvable if this study were to be expanded upon.

The next recommendation that can be made for future studies to improve on these results would be the sample size, at only 129 images total is simply not large enough to be able to train a neural network on the samples, which is the main reason a pre-trained network had to be employed. Compounding this issue is the size of each class, ranging from 59 to just 3; the results bear out that there just aren't enough samples for some classes to be accurately placed. The data obtained states that on average both ResNet versions achieved the same accuracy (70.67%), as the only salient difference between the two networks is the number on convolutional layers it is possible that because this is such a simple problem relatively speaking that the additional complexity and convolution brought in by the 32 extra layers of depth in ResNet-50 is just not necessary for a dataset this small and simple. Although as a counterpoint to this, the inception model has a similar number of layers (48) and outperformed

both ResNet implementations. This is likely due to the differences in the two network's approach to convolutional layers. Another issue is that because the neural networks are a black box, it is not possible to see how the algorithm has got from A to B, only the steps it has taken to get there, at least from the feature data created by this paper. Unfortunately, this is a limitation of the underlying goal of the paper, to study the effect of pre-trained neural nets in combination with different classifiers/filters and as such means that the feature sets are not be human readable.

One issue that is clear from the data, images that have had a Gabor filter applied to them were not classified as well as standard images. There are three possible reasons for this; one is that there was human error in the creation of these images and a more finely tuned set of parameters would result in better classification rates, the second is that by removing the colour the neural networks were deprived of a source of information about the image which could not be overcome by the increased information about the texture of the object. The third possible reason for this low classification rate is simply that these are pre-trained neural nets, and as such have been trained on real colour images and were not used to extracting features from grayscale images. It is likely that a combination of all three lead to the lower accuracy. These could be improved with a more finessed version of the filtering process or training a neural net from scratch on grayscale images, but what is clear is that colour plays a very large role in the classification of pottery sherds and perhaps a hybrid approach could be used to combine colour and texture data.

Finally, there are several ways that this product could be expanded upon and improved. The first would be to increase the amount of data, as mentioned before the sample sizes are too small, and as such have made for a hard time classifying certain sherds. The second would be to expand upon the number of neural networks, classifiers, and filtering techniques to make it a more exhaustive comparison. This was not in scope for this project, however the neural nets themselves could be added as classifiers to evaluate the difference between an all in one neural networks prediction and the more supervised models' prediction, with more classifiers tested for a more rounded comparison. As mentioned earlier colour plays a large role in feature extraction and classification, and any projects using this as a base would be wise to expand the data preparation phase by combining textural data obtained with filtering techniques with colour data from the images.

Lastly, this product is only a script that handles all parts of the data, while this was done in order to keep the scope of the project small, expansions to this work could take the lessons shared above and create a better model for classification, and wrap it all inside a program of mobile app so that the user can take advantage of a model that can accept a photo from a gallery or even from a phone’s camera and classify the pottery sherd instantly in the field.

Chapter 6 – Conclusion

This project set out to provide an overview of the strengths and weaknesses of several methods of identifying and categorising pottery sherds based solely on information found in the image, and to assess whether software could be used to replace a time and effort intensive task of classification. The results obtained indicate that this goal has been achieved, and although there are several factors that could have been improved upon either for this project or for a speculative future project the final conclusion is that this method of analysing and categorising data would be suitable for this purpose so long as the dataset is sufficiently large enough, and recommends using the combination of standard colour images of sherds, the InceptionV3 neural net, and the SVM classifier as a base system, which could be built off of at a later date.

Bibliography

Abadi, M., Khoudeir, M. and Marchand, S. (2012) ‘Gabor Filter-Based Texture Features to Archaeological Ceramic Materials Characterization.’ *In* Elmoataz, A., Mammass, D., Lezoray, O., Nouboud, F., and Aboutajdine, D. (eds) *Image and Signal Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 333–342.

Avidan, S. (2004) ‘Support vector tracking.’ *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(8) pp. 1064–1072.

Burchard, P. (n.d.) ‘Total Variation Geometry I: Concepts and Motivation’ p. 18.

Chetouani, A., Treuillet, S., Exbrayat, M. and Jesset, S. (2020) ‘Classification of engraved pottery sherds mixing deep-learning features by compact bilinear pooling.’ *Pattern Recognition Letters*, 131, March, pp. 1–7.

Eslami, D., Di Angelo, L., Di Stefano, P. and Guardiani, E. (2021) ‘A Semi-Automatic Reconstruction of Archaeological Pottery Fragments from 2D Images Using Wavelet Transformation.’ *Heritage*, 4(1) pp. 76–90.

He, K., Zhang, X., Ren, S. and Sun, J. (2016) ‘Deep Residual Learning for Image Recognition.’ *In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, pp. 770–778.

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2017) ‘ImageNet classification with deep convolutional neural networks.’ *Communications of the ACM*, 60(6) pp. 84–90.

Lowe, D. G. (1999) ‘Object recognition from local scale-invariant features.’ *In Proceedings of the Seventh IEEE International Conference on Computer Vision*. Kerkyra, Greece: IEEE, pp. 1150–1157 vol.2.

Mehrotra, R., Namuduri, K. R. and Ranganathan, N. (1992) ‘Gabor filter-based edge detection.’ *Pattern Recognition*, 25(12) pp. 1479–1494.

Olafenwa, M. (2021) *OlafenwaMoses/ImageAI*. [Online] [Accessed on 19th July 2021]

OpenCV: Introduction (n.d.). [Online] [Accessed on 19th July 2021]

https://docs.opencv.org/4.5.2/d1/dfb/intro.html. [Online] [Accessed on 19th July 2021]

Smith, J. R. and Chang, S.-F. (1996) ‘Tools and techniques for color image retrieval.’ *In* Sethi, I. K. and Jain, R. C. (eds). San Jose, CA, pp. 426–437.

Smith, P., Bespalov, D., Shokoufandeh, A. and Jeppson, P. (2010) ‘Classification of archaeological ceramic fragments using texture and color descriptors.’ *In 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*. San Francisco, CA, USA: IEEE, pp. 49–54.

STL-10 Image Recognition Dataset (n.d.). [Online] [Accessed on 19th July 2021]

https://kaggle.com/jessicali9530/stl10. [Online] [Accessed on 19th July 2021]

tensorflow/tensorflow (2021). tensorflow. [Online] [Accessed on 19th July 2021]

Appendix A

A.1 Feasibility Study

Student name:	John Carter
Student number:	07478311
Degree course:	Computer Science
Supervisor name:	Nicholas Costen
Project title:	Image Texture Analysis and Classification

Course-Specific Learning Outcomes

1. Use knowledge, abilities, and skills for further study and for a range of employment in areas related to scientific and technical computing (MMU, 2020).
2. Interpret legislation appropriate to computer professionals and also be aware of relevant ethical issues and the role of professional bodies (MMU, 2020).
3. Analyse, design, and implement algorithms using a range of appropriate languages and/or methodologies (MMU, 2020).
4. Design, implement and interrogate database systems (MMU, 2020).
5. Apply the principals and operation of Programming languages (MMU, 2020).

6. Demonstrate effective decision making and creative problem-solving skills, and identify appropriate practices within a professional, legal, and ethical framework (MMU, 2020).
7. Critically appraise and apply suitable artificial intelligence techniques for a variety of software systems (MMU, 2020).

Project Background

Certain natural categories of object (for example types of rock) lack natural shapes or indicative colours. However, they can be distinguished visually on the basis of the distribution and nature of features of a significantly smaller scale than the object itself, the texture of the surface (MMU, 2020).

There are several mathematical techniques to accomplish this. Tuceryan and Jain (1993) suggested four main categories: statistical, geometrical, model based, and signal processing, which can again be further subdivided to suit the needs of the application.

This project is to create a product that can use these techniques to analyse the texture of archaeological pottery, to build off the current manual process which involves an expert looking at several factors including but not limited to the colour of the clay and any imperfections mixed into the clay (which is determined using the number of imperfections, size, distribution, and shape) (MMU, 2020).
The product will use the different techniques described above to create algorithms to measure these features, and assess the performant statistics of each algorithm, i.e. the speed and accuracy.

Aim

The aim of this project is to create software that will implement several different algorithms that will process an image texture (in this case pottery), analyse the properties to classify it and compare the different methods through a desktop application.

Objectives

The objectives for this task have been broken up into logical chunks to help with the overall velocity of the project, although these tasks may overlap/be completed in a different order:

- Research the algorithms needed to write this program, using previous studies/ in this field (using the library or an online scientific paper aggregator like Google scholar).
- Perform a theoretical analysis on the algorithms expected performance.
- Research the best technologies to fit the nature of the project, for front-end, back-end, and likely a database/data storage solution.
- Implement a basic program that will accept and store the required data, an interface, and framework for the algorithms.
- Implement each of the algorithms into the program and make sure they work as intended.
- Testing and final polish of the program.
- Capture data across a large enough data set on the finished program.
- Summarise the results and present the findings in a final report, comparing the algorithms performant results, and compare my implementation to earlier comparison, and possibly other studies in the same area.

Problems

One stumbling block that could occur is in the algorithms themselves, after an initial investigation into the algorithms needed for the project the complexity is currently above my understanding. This is something that can/will be mitigated by keeping in contact with the project supervisor in the process of research/coding, and early iteration on the coding side, in order to keep momentum by making small adjustments and getting feedback on those iterations. There will also be the AI block which will help with more complex algorithms.

Another issue to consider is that when implemented my algorithms may not work as intended, causing the data obtained to be unusable for the final report/analysis, or there may be an issue with the main programs' ability. Again, this is an issue that can be mitigated with lots of iteration and communication with the project supervisor.

The final problem that could arise is time management, as lots of the project will be scheduled out in advance, but of course workloads/deadlines can shift over a products lifecycle. This will be mitigated by trying to keep on sprint, the method chosen to keep the project manageable,

but also reassessing when/how long a sprint will take if any difficulties arise, while also keeping the project supervisor informed as to any potential/actual problems that are stalling the project.

Required Resources

- Development Computer
- Text editor (MS Word / LaTeX editor)
- IDE (Visual studio 2017/Code)
- SQL database/Python/C#/ dev tools
- MS Teams
- GitHub/Git
- Data of Pottery samples for analysis

Schedule

This project will be divided into sprints, with a normal sprint taking two weeks, but this will be subject to change based on the complexity of the task. Some tasks will be broken into sprints later.

- Sprint 1: Feasibility Study/initial research | 4 weeks/2 sprints
- Sprint 2: Algorithm research | 4 weeks/2 sprints
- Sprint 3: Initial Algorithm analysis | 2 weeks/1 sprint
- Sprint 3: Technology research | 1 week/ ½ sprint
- Sprint 4: Project prototype (sans algorithms) | 2 weeks/1 sprint
- Sprint 5: Algorithm implementation/refining | 4 weeks (to be broken up into sprints after more understanding)
- Sprint 6: Bug/usability testing | 2 weeks/1 sprint (subject to change/bugs will be dealt with as they crop up)
- Sprint 7: Data collection | 1 week/ ½ sprint
- Sprint 8: Data Analysis | 2 weeks/1 sprint
- Sprint 9: Final report | 2 weeks/1 sprints

Ethics number

26729

References

1-7. MMU. 2020. Project Handbook. [Online] Available at: https://moodle.mmu.ac.uk/pluginfile.php/4314541/mod_resource/content/10/6Z1101_Handbook_2020_v1.pdf [Accessed 21 October 2020].

8. MMU. 2020. NPC.03 | SCMDT Project Selection. [online] Available at: http://projectlist.cmdt-students.net/?q=content/npc03 [Accessed 21 October 2020].

Appendix B

A.1 Prototype

```
% prepare and split 70/30
dStore =
imageDatastore('Prototype_Samples','IncludeSubfolders',true,'LabelSource','fo
ldernames');
[trainSet,testSet] = splitEachLabel(dStore,0.7,'randomized');

% display images from training set
figure;
numTrainImages = numel(trainSet.Labels);
idx = randperm(numTrainImages,16);
figure
for i = 1:16
    subplot(4,4,i)
    I = readimage(trainSet,idx(i));
    imshow(I)
end

% load neural net
net = resnet18;

% resize dataset
inputSize = net.Layers(1).InputSize;
augmentTrain = augmentedImageDatastore(inputSize, trainSet);
augmentTest = augmentedImageDatastore(inputSize, testSet);
```

```
% feature extraction
layer = 'pool5';
trainFeatures = activations(net, augmentTrain, layer, 'OutputAs', 'rows');
testFeatures = activations(net, augmentTest, layer, 'OutputAs', 'rows');
%label extraction
trainLabels = trainSet.Labels;
testLabels = testSet.Labels;

% classify and predict
classifier = fitcecoc(trainFeatures, trainLabels);
potPredict = predict(classifier, testFeatures);

% calculate acuracy
accuracy = mean(potPredict == testLabels);
disp(accuracy);
figure;
protoConfusion = confusionchart(testLabels, potPredict);
protoConfusion.Title = 'Prototype sample images with ResNet-18 CNN and SVM
classifier';
```

A.2 Gabor Filter

```
%Var initialise
srcFolder = 'Samples\CM\';
filePattern = fullfile(srcFolder, '*jpg');
files = dir(filePattern);
dstFolder = "Gabor_Samples\CM\";
wavelength = [2,4,6];
orientation = [0,45,90,135];
gabArray = gabor(wavelength, orientation);

%loop applies gabor filter and writes to file
for i = 1:length(files)
    baseFileName = files(i).name;
    fullFileName = fullfile(srcFolder, baseFileName);
    dstFullFile = dstFolder + baseFileName;
    img = imread(fullFileName);
    img = rgb2gray(img);
    gabMag = imgaborfilt(img, gabArray);

    [height, width, numFilters] = size(gabMag(:,:,:));
    combineMag = zeros(height,width);

    % Average the values from each filter
    for j = 1:height
        for k = 1:width
            sum = 0;
            for f = 1:numFilters
                sum = sum + gabMag(j,k,f);
            end
            combineMag(j,k) = sum / numFilters;
        end
    end

    % concat three duplicate layers to spoof RGB channels on feature
```

```
    % extraction
    grayImage = combineMag;
    finalImage = cat(3,grayImage, grayImage, grayImage);
    % write filtered image to disk
    imwrite(uint8(finalImage), dstFullFile);
end

% preview of image filter result
figure;
imshow(dstFullFile);
title('Pottery sherd after Gabor image filter applied', 'FontSize',16);
```

A.3 Project

```
% No filter (standard) prep and split
stdDStore = imageDatastore('Samples', 'IncludeSubfolders',true,
'LabelSource',"foldernames");
[stdTrainSet, stdTestSet] = splitEachLabel(stdDStore,0.7,'randomized');

% Gabor prep and split
gbrDStore =
imageDatastore('Gabor_Samples',"IncludeSubfolders",true,"LabelSource","folder
names");
[gbrTrainSet, gbrTestSet] = splitEachLabel(gbrDStore,0.7,'randomized');
```

```
% ResNet18 feature extract
net = resnet18;
inputSize = net.Layers(1).InputSize;
analyzeNetwork(net);

% resizing standard images
stdR18AugTrain = augmentedImageDatastore(inputSize,stdTrainSet);
stdR18AugTest = augmentedImageDatastore(inputSize,stdTestSet);
% gabor resize
gbrR18AugTrain = augmentedImageDatastore(inputSize, gbrTrainSet);
gbrR18AugTest = augmentedImageDatastore(inputSize,gbrTestSet);

% standard feature extraction & writing datapoints to file
layer = 'pool5';
stdR18TrainFeat = activations(net,stdR18AugTrain, layer, 'OutputAs',"rows");
writematrix(stdR18TrainFeat, 'Datapoints\stdR18Train.xlsx');
stdR18TestFeat = activations(net, stdR18AugTest, layer, 'OutputAs',"rows");
writematrix(stdR18TrainFeat, 'Datapoints\stdR18Test.xlsx');
%Gabor extraction
gbrR18TrainFeat = activations(net,gbrR18AugTrain,layer,'OutputAs',"rows");
writematrix(stdR18TrainFeat, 'Datapoints\gbrR18Train.xlsx');
gbrR18TestFeat = activations(net, gbrR18AugTest,layer,'OutputAs',"rows");
```

```
writematrix(stdR18TrainFeat, 'Datapoints\gbrR18Test.xlsx');

%standard label extraction
stdR18TrainLabels = stdTrainSet.Labels;
stdR18TestLabels = stdTestSet.Labels;
%Gabor label extraction
gbrR18TrainLabels = gbrTrainSet.Labels;
gbrR18TestLabels = gbrTestSet.Labels;
```

```
% ResNet50 feature extract
net = resnet50;
inputSize = net.Layers(1).InputSize;

% resizing standard images
stdR50AugTrain = augmentedImageDatastore(inputSize,stdTrainSet);
stdR50AugTest = augmentedImageDatastore(inputSize,stdTestSet);
% gabor resize
gbrR50AugTrain = augmentedImageDatastore(inputSize, gbrTrainSet);
gbrR50AugTest = augmentedImageDatastore(inputSize,gbrTestSet);

% standard feature extraction
layer = 'fc1000';
stdR50TrainFeat = activations(net,stdR50AugTrain, layer, 'OutputAs',"rows");
writematrix(stdR18TrainFeat, 'Datapoints\stdR50Train.xlsx');
stdR50TestFeat = activations(net, stdR50AugTest, layer, 'OutputAs',"rows");
writematrix(stdR18TrainFeat, 'Datapoints\stdR50Test.xlsx');
% Gabor extraction
gbrR50TrainFeat = activations(net,gbrR50AugTrain,layer,'OutputAs',"rows");
writematrix(stdR18TrainFeat, 'Datapoints\gbrR50Train.xlsx');
gbrR50TestFeat = activations(net, gbrR50AugTest,layer,'OutputAs',"rows");
writematrix(stdR18TrainFeat, 'Datapoints\gbrR50Test.xlsx');

% standard label extraction
stdR50TrainLabels = stdTrainSet.Labels;
stdR50TestLabels = stdTestSet.Labels;
```

```
% Gabor label extraction
gbrR50TrainLabels = gbrTrainSet.Labels;
gbrR50TestLabels = gbrTestSet.Labels;

% InceptionV3 feature extract
net = inceptionv3;
inputSize = net.Layers(1).InputSize;
analyzeNetwork(net);

% resizing standard images
stdIV3AugTrain = augmentedImageDatastore(inputSize,stdTrainSet);
stdIV3AugTest = augmentedImageDatastore(inputSize,stdTestSet);
% Gabor resize
gbrIV3AugTrain = augmentedImageDatastore(inputSize,gbrTrainSet);
gbrIV3AugTest = augmentedImageDatastore(inputSize,gbrTestSet);

% standard feature extraction
layer = 'avg_pool';
stdIV3TrainFeat = activations(net, stdIV3AugTrain,layer,'OutputAs',"rows");
writematrix(stdR18TrainFeat, 'Datapoints\stdIV3Train.xlsx');
stdIV3TestFeat  = activations(net,stdIV3AugTest,layer,'OutputAs',"rows");
writematrix(stdR18TrainFeat, 'Datapoints\stdIV3Test.xlsx');
% Gabor extraction
gbrIV3TrainFeat = activations(net,gbrIV3AugTrain,layer,'OutputAs','rows');
writematrix(stdR18TrainFeat, 'Datapoints\gbrIV3Train.xlsx');
gbrIV3TestFeat = activations(net,gbrIV3AugTest,layer,'OutputAs','rows');
writematrix(stdR18TrainFeat, 'Datapoints\gbrIV3Test.xlsx');

% standard label extraction
stdIV3TrainLabels = stdTrainSet.Labels;
stdIV3TestLabels = stdTestSet.Labels;
% Gabor label extraction
gbrIV3TrainLabels = gbrTrainSet.Labels;
gbrIV3TestLabels = gbrTestSet.Labels;
```

```
% K-NN classifier (all models)
% standard R-18
stdR18KNNClassifier = fitcknn(stdR18TrainFeat, stdR18TrainLabels,
'NumNeighbors', 1);
stdR18KNNPredictor = predict(stdR18KNNClassifier, stdR18TestFeat);
% Gabor R-18
gbrR18KNNClassifier = fitcknn(gbrR18TrainFeat, gbrR18TrainLabels,
'NumNeighbors',1);
gbrR18KNNPredictor = predict(gbrR18KNNClassifier, gbrR18TestFeat);
% standard R-50
stdR50KNNClassifier = fitcknn(stdR50TrainFeat, stdR50TrainLabels,
'NumNeighbors', 1);
stdR50KNNPredictor = predict(stdR50KNNClassifier, stdR50TestFeat);
% Gabor R-50
gbrR50KNNClassifier = fitcknn(gbrR50TrainFeat, gbrR50TrainLabels,
'NumNeighbors', 1);
gbrR50KNNPredictor = predict(gbrR50KNNClassifier, gbrR50TestFeat);
% standard IV3
stdIV3KNNClassifier = fitcknn(stdIV3TrainFeat, stdIV3TrainLabels,
'NumNeighbors', 1);
stdIV3KNNPredictor = predict(stdIV3KNNClassifier, stdIV3TestFeat);
% Gabor IV3
gbrIV3KNNClassifier = fitcknn(gbrIV3TrainFeat, gbrIV3TrainLabels,
'NumNeighbors', 1);
gbrIV3KNNPredictor = predict(gbrIV3KNNClassifier, gbrIV3TestFeat);

% results
stdR18KNNAccuracy = mean(stdR18KNNPredictor == stdR18TestLabels);
gbrR18KNNAccuracy = mean(gbrR18KNNPredictor == gbrR18TestLabels);
stdR50KNNAccuracy = mean(stdR50KNNPredictor == stdR50TestLabels);
gbrR50KNNAccuracy = mean(gbrR50KNNPredictor == gbrR50TestLabels);
stdIV3KNNAccuracy = mean(stdIV3KNNPredictor == stdIV3TestLabels);
gbrIV3KNNAccuracy = mean(gbrIV3KNNPredictor == gbrIV3TestLabels);
disp(stdR18KNNAccuracy);
disp(gbrR18KNNAccuracy);
```

```
disp(stdR50KNNAccuracy);
disp(gbrR50KNNAccuracy);
disp(stdIV3KNNAccuracy);
disp(gbrIV3KNNAccuracy);
```

```
% SVM classifier (all models)
% standard R-18
stdR18SVMClassifier = fitcecoc(stdR18TrainFeat, stdR18TrainLabels);
stdR18SVMPredictor = predict(stdR18SVMClassifier, stdR18TestFeat);
% Gabor R-18
gbrR18SVMClassifier = fitcecoc(gbrR18TrainFeat, gbrR18TrainLabels);
gbrR18SVMPredictor = predict(gbrR18SVMClassifier, gbrR18TestFeat);
% standard R-50
stdR50SVMClassifier = fitcecoc(stdR50TrainFeat, stdR50TrainLabels);
stdR50SVMPredictor = predict(stdR50SVMClassifier, stdR50TestFeat);
% Gabor R-50
gbrR50SVMClassifier = fitcecoc(gbrR50TrainFeat, gbrR50TrainLabels);
gbrR50SVMPredictor = predict(gbrR50SVMClassifier, gbrR50TestFeat);
% standard IV3
stdIV3SVMClassifier = fitcecoc(stdIV3TrainFeat, stdIV3TrainLabels);
stdIV3SVMPredictor = predict(stdIV3SVMClassifier, stdIV3TestFeat);
% Gabor IV3
gbrIV3SVMClassifier = fitcecoc(gbrIV3TrainFeat, gbrIV3TrainLabels);
gbrIV3SVMPredictor = predict(gbrIV3SVMClassifier, gbrIV3TestFeat);
```

```
% results
stdR18SVMAccuracy = mean(stdR18SVMPredictor == stdR18TestLabels);
gbrR18SVMAccuracy = mean(gbrR18SVMPredictor == gbrR18TestLabels);
stdR50SVMAccuracy = mean(stdR50SVMPredictor == stdR50TestLabels);
gbrR50SVMAccuracy = mean(gbrR50SVMPredictor == gbrR50TestLabels);
stdIV3SVMAccuracy = mean(stdIV3SVMPredictor == stdIV3TestLabels);
gbrIV3SVMAccuracy = mean(gbrIV3SVMPredictor == gbrIV3TestLabels);
disp(stdR18SVMAccuracy);
disp(gbrR18SVMAccuracy);
disp(stdR50SVMAccuracy);
disp(gbrR50SVMAccuracy);
disp(stdIV3SVMAccuracy);
disp(gbrIV3SVMAccuracy);
```

```
% accuracy comparisons
% confusions matrices for all classifiers
figure;
% standard KNN confusions
stdR18KNNConfusion = confusionchart(stdR18TestLabels, stdR18KNNPredictor);
stdR18KNNConfusion.Title = 'Unfiltered images with ResNet-18 CNN and K-NN
classifier';
stdR50KNNConfusion = confusionchart(stdR50TestLabels, stdR50KNNPredictor);
stdR50KNNConfusion.Title = 'Unfiltered images with ResNet-50 CNN and K-NN
classifier';
stdIV3KNNConfusion = confusionchart(stdIV3TestLabels, stdIV3KNNPredictor);
stdIV3KNNConfusion.Title = 'Unfiltered images with InceptionV3 CNN and K-NN
classifier';
% standard SVM confusions
stdR18SVMConfusion = confusionchart(stdR18TestLabels, stdR18SVMPredictor);
stdR18SVMConfusion.Title = 'Unfiltered images with ResNet-18 CNN and SVM
classifier';
stdR50SVMConfusion = confusionchart(stdR50TestLabels, stdR50SVMPredictor);
stdR50SVMConfusion.Title = 'Unfiltered images with ResNet-50 CNN and SVM
classifier';
stdIV3SVMConfusion = confusionchart(stdIV3TestLabels, stdIV3SVMPredictor);
```

```
stdIV3SVMConfusion.Title = 'Unfiltered images with InceptionV3 CNN and SVM
classifier';
% Gabor KNN confusions
gbrR18KNNConfusions = confusionchart(gbrR18TestLabels, gbrR18KNNPredictor);
gbrR18KNNConfusions.Title = 'Gabor images with ResNet-18 CNN and K-NN
classifier';
gbrR50KNNConfusions = confusionchart(gbrR50TestLabels, gbrR50KNNPredictor);
gbrR50KNNConfusions.Title = 'Gabor images with ResNet-50 CNN and K-NN
classifier';
gbrIV3KNNConfusions = confusionchart(gbrIV3TestLabels, gbrIV3KNNPredictor);
gbrIV3KNNConfusions.Title = 'Gabor images with InceptionV3 CNN and K-NN
classifier';

% Gabor SVM confusions
gbrR18SVMConfusions = confusionchart(gbrR18TestLabels, gbrR18SVMPredictor);
gbrR18SVMConfusions.Title = 'Gabor images with ResNet-18 CNN and SVM
classifier';
gbrR50SVMConfusions = confusionchart(gbrR50TestLabels, gbrR50SVMPredictor);
gbrR50SVMConfusions.Title = 'Gabor images with ResNet-50 CNN and SVM
classifier';
gbrIV3SVMConfusions = confusionchart(gbrIV3TestLabels, gbrIV3SVMPredictor);
gbrIV3SVMConfusions.Title = 'Gabor images with InceptionV3 CNN and SVM
classifier';
```