Classifying The Architectural Style of Catalan Buildings Using Both Deep Learning and Non-Deep Learning Methods

Course Project - Advanced Machine Learning

Àlex Martorell i Locascio
Louis Van Langendonck

Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya
23/12/2022

Contents

Abstract

In this study, a Catalan Architecture image classification model was developed using both deep learning and non-deep learning methods. The multiclass classification problem involves the eight most frequently occuring styles in Catalunya. The deep learning approach considered fine tuning a convolutional neural network. The non-deep learning approach encompasses extracting features from the images and training traditional machine learning models on the extracted features. The results showed that the deep learning methods outperformed the non-deep learning ones. However, the final model shows room of improvement, with as main limiting factors the data quality and computational costs.

1 Problem Description

Catalunya has a rich architectural heritage, containing buildings ranging from Romanesque (from about the 8th century on) to contemporary styles [1]. The Generalitat de Catalunya hosts a repository of noteworthy monuments and buildings, with each entry containing information like name, adress, state of conservation and the associated architectural style(s), often supplemented by one or more images [2]. However, the architectural style labeling problem is a non-exact science heavily reliant on varying conventions. Typically, a building contains elements of different style categories and similarly, each of these categories shares characteristics with others (like for example each "neo-" style that as per definition reinterprets and re-uses the trends from a certain architectural style from the past). To enrich this discussion, this project aims to research the ability of Machine Learning algorithms to differentiate between prominent architectural styles of Catalan buildings, trained on images provided by the repository of The Generalitat de Catalunya.

To construct the dataset, a web-scraper is build (`1_1_data_scraper.ipynb`) that browses through the aforementioned website and downloads images of each respective style. Of the different style labels provided by the site, the top eight most prominent and often occurring style families are considered: Baroque, Contemporary, Gothic, Modernism, Neoclassicism, Noucentisme, Renaissance and Romanesque. It is chosen to only download images of buildings that have one label assigned to them. This decreases, but surely not excludes, the amount of structures containing elements of multiple different styles. 15428 images constitute the data set that is used to train and test the architectural style multi-class classifier.

2 Previous Work

It is well known that Image classification is an often occuring problem in Machine Learning nowadays. However, this field is not constrained to only modern methods. In [3], (1999) there is an approximation to the problem by building a color histogram of the image. The authors use SVM classifiers with an RBF kernel and obtain good results on some datasets . Specifically, hyperspectral image classification (see [4]) has been a hot topic, and have used kernel design to increase accuracy with respect to training an SVM using a single kernel.

However, this is not only computationally very expensive, but will lead to poor results when the data set increases in complexity. Convolutional Neural Networks have provided the framework for extracting feature vectors from images that can later be trained on an SVM or an MLP. A review of the techniques used for image classification can be found in [5].

In the architectural context, [6] tries to classifiy current tendencies in real estate buildings in China, based on images of the facades of such buildings. It manages to integrate economic factors into the final predictive model.

Recently, Deep Learning has shown to be most successful in such tasks. Some authors have applied it to architectural style classification as well like for example in [7]. It also provides a survey of several papers that cover classification of visual elements in art.

3 Data Exploration

3.1 Train-test split

To estimate the generalization error of the final model, a collection of images (called the test set) must be put aside, strictly invisible to the model during training. In order to train and test on datasets representing the class proportions in the repository, it is chosen to split in a stratified manner. However, note that in the data scraping method used, a single building can have more than one image in the data set. Although by exploring the data, it is clear that these tend to contain vastly different parts of the building, for the train-test split, it is considered a good practice to keep different images of the same building separated between the splits. This way, any information leakage to the test set is avoided such that the test data can truly be considered 'unseen'. To do this, in `1_2_train_test_split.ipynb`, manual stratified splitting is done per building instead of per image. This results in an per-image split that is close to, but not exactly 80% - 20%, a small difference that should not impact generalization error estimation. The train set now contains 12291 images, while the test set contains 3135 images. Note that all decisions or analysis from now on will be based on information coming from the training set. This prevents any form of data leakage to the test set. It is important to note that any step containing a controllable random component, like train-test splitting here, has a specified random state in the code to support reproducability.

3.2 Pre-processing

Image pre-processing will differ based on the model used. The pre-processing steps used for the different feature extraction techniques are each described in Section 3.5, which covers pre-processing for the methods described in Section 5.1. Pre-processing specific to models covered in Section 5.2 are specified at that point.

3.3 Exploratory Statistics

Most importantly, the class distribution is looked into. A plot of the image counts of each style is displayed in Figure 3.1. It is clear that the data set is significantly imbalanced. Mainly the Renaissance-class could be underrepresented, especially for Convolutional Neural Network based methods (Section 5.2), as these methods typically continuously keep improving proportionally to amount of images provided [8]. Clearly, this class imbalance will have to be taken into account in model training.

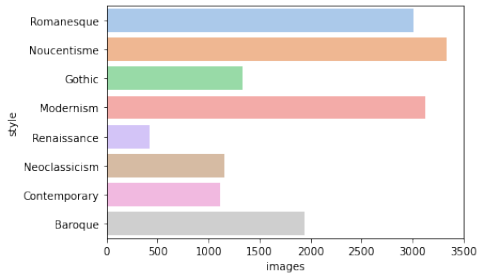

Figure 3.1: Distribution of classes in training set.

Finally, to get an idea of the size of the images, a histogram is plotted in Figure 3.2, displaying the distribution of the pixel sizes of the images (so pixel-width $\times$ pixel-height). Although there are clearly high resolution outliers present, each method considered in the paper will resize all images to computationally manageable sizes. Therefore, this distribution is disregarded from now on.

3.4 Data Quality

Upon investigation of the data set, a few things become clear about the quality of the data and the complexity of the classification problem:

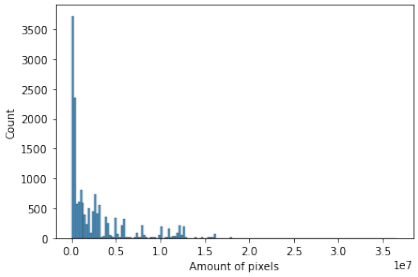

Figure 3.2: Distribution of image size in pixels across training set.

- The subject of the image can vary greatly. Because architectural buildings cover a wide range of structures, each with its own expression of the corresponding style (churches, houses, hospitals, squares, office blocks, etc.), the complexity of the classification problem can be considered high.
- A significant amount of images seem unclear, for different possible reasons: multiple buildings are displayed where only one of them is the actual subject of the image, the building contains elements of different styles (not mentioned in the labeling) or the building is in a poor conservational state.
- The labeling performed by the Generalitat de Catalunya does not seem consistent and/or prone to mistakes. Given there is a lot of ambiguity in agreeing on the architectural style of a building, it is often easy to question the labeling supplied by the repository. It is important to note that the biases from the Generalitat de Catalunya will probably leak into the models.
- A fairly balanced mix of grayscale and colored images is present in the data set. This will only be of importance in Section 3.5.2, which covers the feature extraction method that does not make use of Convolutional Neural Networks. Every other method does make use of convolutional neural networks which works on both color and grayscale images.

3.5 Feature Extraction

A major proportion of the models to build make use of an explicit feature extraction step. There are however, multiple ways of going from an image to an informative vector. Two methods are explored here, one with and one without the use of pre-trained deep convolutional neural networks.

3.5.1 Pre-trained Networks

Using pre-trained Convolutional Neural Networks (CNNs) is a well-established method of obtaining image features. These networks are obviously not trained on architectural style classification but rather on millions of images of more than 20000 general categories like for example 'strawberry' or 'balloon' [9]. However, by stripping these rich architectures from their top fully-connected layers, the output changes from a soft-max vector to a more general feature vector representing latent concepts present in the image. Four pre-trained networks are chosen for this task based on their established performance: Xception [10], VGG19 [11], Inception-ResNet-v2 [12] and EfficientNetB7 [13].

The feature extraction process can be found in `2_1_feature_extractor_cnn.ipynb` and broadly goes through the following steps for each of the networks mentioned above:

1. The image is resized and scaled according to the image preprocessing on which the pretrained models are trained. For all models, this means resizing to (299,299) and (-1, 1)-scaling.
2. All image tensors are now ran through the model and the resulting feature vectors stored for model training.

3.5.2 Visual Bag Of Words using SIFT

This is a non deep-learning method that builds a Visual Bag Of Words (VBOW) using the Scale-Invariant Feature Tranform algorithm (SIFT) [14]. SIFT is an algorithm that for an image it detects

an unspecified amount of interesting local keypoints (called 'patches') and describes them in a vector of length 128. The keypoint detection is designed to be scale-, translation- and rotation-invariant, which is one of the reasons it is generally the highest performing keypoint detection algorithm [15]. The patch/keypoint detection technique is build on the concept of Gaussian scale-space. This space can be considered as an embedding of the original image into a space of derived images by continuous convolution of a Gaussian Kernel with varying variance (and mean zero). More intuitively, each of these derived images in the scale-space represents a loss of detail for a particular zoom (parametrized by the variance of the Gaussian). Afterwords a Difference of Gaussians (DoG) Operator is applied on this space as this smooths out homogeneous and uninteresting points of the image. Finally, after applying the DoG, extremas are computed and associated back to the input image, described in a vector of length 128 and used as keypoints. A more detailed explanation of the inner working of the SIFT algorithm can be found in [16].

These keypoints in different images can be compared and 'matched' if it represents a similar object or detail. An example is displayed in Figure 3.4, where two images labeled as Romanesque are loaded in, keypoints found and matched. However, this matching does not deliver a fixed-length feature vector. To get there, a Visual Bag Of Words (VBOW) is build by adapting a method described in [17]. In `2_2_feature_extractor_no_cnn`, the following steps are executed:

1. Each image is rescaled to (299,299) to lower the computational cost of the following steps. Subsequently it is converted into grayscale as SIFT considers only a monochrome intensity image [14], hence losing color information. For each of these, all the keypoints/feature patches are extracted and assembled in a list without keeping track of image or labeling.

2. On this set, the k-means algorithm is applied in order to cluster patches that have similar representations. These clusters can hence be considered latent concepts. This process is repeated for multiple k's. The respective inertias are plotted in Figure 3.3. From this plot the choice of k is not obvious. The elbow-method would probably suggest $k = 800$. However, the inertia clearly keeps lowering with higher k such that higher dimensional options might be preferable. Too high dimensional data on the other hand is computationally costly. Therefore, the rather arbitrary choice of $k = 1600$ is made, eventually resulting in 1600-dimensional feature vectors. However, no dramatic changes are expected in feature performance when building models.

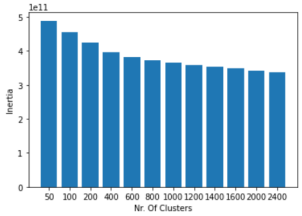

Figure 3.3: Inertia plot of varying K for K-means clustering of keypoints.

3. Finally, for each image, each of its patches are assigned to one of the k clusters by finding the closest cluster center. Using this, a k-dimensional vector is created in which each position keeps track of how many patches belong to that cluster for that image. After normalizing, the final feature vector is obtained. This is done for each image and stored for model training.

4 Re-sampling and Validation Strategy

As already mentioned, a stratified test set is created containing 20% of the data. The remaining training images are now again splitted into a stratified 80% - 20% train- and validation set used for model selection. As there are 9832 images left in the training set after this split, the data set is considered large enough to warrant not using cross-validation for parameter optimization and model selection. Note that the models are trained on large, high-dimensional data and can thus be

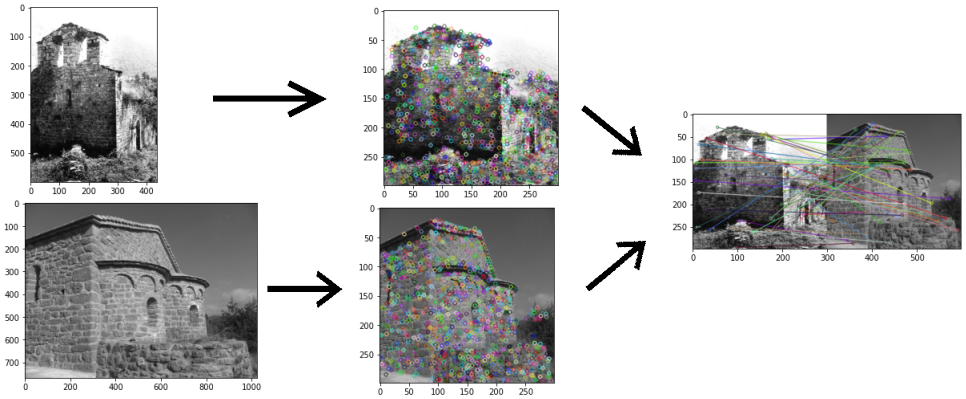

Figure 3.4: Example of SIFT keypoint detection (middle) and matching (right).

considered computationally costly. Therefore, not using cross-validation allows for increasing the hypothesis space.

The metric of choice for parameter optimization and model selection is the weighted average F1-score defined as $\sum_c F1_c \frac{n_c}{n_{tot}}$, with c a category, n_c the amount of images of that category, n_{tot} the total amount of images and the F1-score defined as $F1 = \frac{TP}{TP+\frac{1}{2}(FP+FN)}$, with TP the amount of true positives, FP the amount of false positives and FN the amount of false negatives. The F1-score for a single category is a measure that balances both recall and precision, hence preferred above accuracy in imbalanced classification. A weighted average is preferred above an overall average because the class imbalance in the dataset represents the actual imbalance in architectural styles present in Catalunya (fe. It is not expected to find many Renaissance buildings in Catalunya). As the goal of the project is to be able to classify as accurately as possible buildings present in Catalunya, it makes sense to add more value to correctly classifying the styles that occur often.

The model with the best weighted F1-score on the validation set is chosen as final model. It is subsequently trained on the entire training set (including the prior validation set) and its generalization error estimated on the test set.

5 Model Training

There is no single way of building an image classifier. Hence, different potentially interesting and/or powerful methods are looked into and optimized. The methods used for this task can be broadly classified in two families: First, methods that use as input, feature vectors that are extracted independently from the model applied on it. The corresponding Models used are described in Section 5.1. The second family of models are Convolutional Neural Networks (CNN) where feature extraction happens implicitly and the inputs are the immediate image tensors. The corresponding models are covered in Section 5.2.

5.1 Feature Extraction Based Methods

Using the feature extraction strategies explained in Section 3.5, five feature vectors are available per image. Table 5.1 lists the dimension of such vectors. These can be used alone or combined by concatenating the vectors. These serve as input to a classifier, like in this case: A Support Vecort Machine (SVM) or a Multilayer Perceptron Neural Network (MLP).

Feature	*Xception*	*EfficientNetB7*	*VGG19*	*Inception_Resnet_v2*	*SIFT*
Dimensionality	2048	2560	512	1536	1600

Table 5.1: Dimensionalty of feature extracted vectors

5.1.1 Support Vector Machines (SVM) and Kernel Methods

Note that all the code for this section can be found in notebook `3_1_SVM_training.ipynb`

Since their first appearance, many formulations of the Support Vector Machine problem have been proposed [18]. Initially developed as a binary classifier, in the 1990s SVM formulations for multi-class classification problems are developed, mainly **One-versus-one** (OvO) and **One-versus-rest** (OvR). These modifications of the original SVM classifier are necessary for the image classification task that is being dealt with. The notation used in this section is as follows: Given an feature vector x^i, the goal is to assign it a label $y^i \in \{1, 2, 3, 4, 5, 6, 7, 8\}$, representing the different art periods.

Multiclass SVMs. Briefly, **One-versus-rest** builds c binary SVM classifiers where c is the number of classes (i.e. c hyperplanes that intersect in $\mathbb{R}^d$). Each of these classifiers separates one class from the rest. Observe that c decision functions are obtained while learning, and an unseen validation point (x', y') is evaluated on these c functions (i.e. $y'_k = w_k^T x' + b_k \; \forall k = 1 \div c$). Since x' could belong to several regions defined by the hyperplanes, the maximum decision function value is chosen (i.e. $y'_k = \arg\max_j w_k^T x' + b_k$.)
In **One-versus-one**, a binary SVM classifier is built for each pair of classes. Note that with c classes, $\binom{c}{2} = \frac{c(c-1)}{2}$ classifiers are trained. Then, given an x' validation set point, the class label is assigned by majority voting (the class is the one where x' has scored "+1" more times).

As any Machine Learning algorithm, there are its drawbacks to take into consideration. OvR builds c classifiers but trained on different tasks (1-vs-rest, 2-vs-rest...), so y'_c value could be biased (on different scales). Another problem that training sets are imbalanced.

As it has been said, both approaches OvR and OvO can lead to ambiguous regions. These are parts of the feature space that do not belong to any decision region. For a large c, OvO incurs in more computation time because it has more classifiers to train, and evaluating test points is also more costly. [19]. There are evolutions and improvements to the SVM formulation, however, this report is restricted to the ones implemented in Python for usability purposes.

Loss functions. The loss function for the SVM is defined as $L(y) = \max(0, 1 - t \cdot y)$, where $t = \pm 1$ are the actual labels and y is the prediction. In SVM this function (L1) or the square of the hinge loss (L2) are used as loss functions. The penalty for the regularization can either be the $\|\cdot\|_1$ (RL1) and $\|\cdot\|_2$ norm (RL2).

SVMs in Python. In this report, the Python implementations from scikit-learn are used, which are `liblinear` and `libsvm`. [20]. The first one is well known for its lower computation cost with respect to the latter one. `libsvm` via the `SVC()` function solves the dual problem (in most cases) via the SMO algorithm and accepts any kernel function. `liblinear` does not support the kernel trick. In other words, `liblinear` is used only when the kernel is linear ($k(x^i, x^j) = \langle x^i, x^j \rangle$). In Python the liblinear framework to solve SVM classifications the function is called`LinearSVC()` [20, 21, 22].

`SVC()` implements OvO approach. In LinearSVC(), it is possible to do train 3 formulations of the SVM problem: L1RL2, L2RL1, L2RL2. Table 5.2 displays the differences in accuracy between these formulations. It is clear that in most cases the default option (Square of the hinge loss) and square norm for penalization is the best option.

Formulation	L2RL2 (default)	L1RL2	L2RL1
Weighted average	54.30 %	53.56 %	53.81 %

Table 5.2: Accuracy on the Inception_Resnet_v2 vector with a Linear SVM (best parameters found for each combination)

Class balance in SVMs. Both `SVC()` and `LinearSVC()` support class weights for unbalanced data sets. Setting `class_weight= 'balanced'` will adjust the weights of each class by weighing the

hyper-parameter C:

$$C_k = w_k \cdot C \quad w_k = \frac{n}{m \cdot n_k}$$

where n_k is the number of training data points in class k. m is the total number of classes. Observe that the only change in the formulation is the importance of a misclassified data point, which now is different per class. Table 5.3 shows not only the difference if the class balance is taken into account but also the importance of choosing the correct score to measure the accuracy of the model.

Formulation	No class balance	Class balance
Micro F1	54.98 %	54.65 %
Weighted Average	53.41 %	54.30 %

Table 5.3: SVM class balance tested on Inception_Resnet_v2 feature vector, (Linear SVM $C = 0.01$)

Since is this a unbalanced dataset, it is reasonable to penalize minority class misclassification more severely. The table above shows the significant difference in the weighted score taking the class balance parameter.

Also, if there is no class balance, taking a look at the classification matrix it can be seen that classes with few images are completely disregarded, yielding a class F1-score near to zero. Without class balance, overall accuracy is favored above per class accuracy, as shown in the table (higher Micro F1).

5.1.1.1 Standard kernels

The regularization SVM problem is well known to be:

$$\min_{f \in \mathcal{H}} \left\{ \frac{1}{n} L_{hinge}\big(f(x^i,)y^i\big) + \lambda \|f\|^2_{\mathcal{H}} \right\} \tag{5.1}$$

By the Representer Theorem [23]:

$$\hat{f}(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x) = \sum_{i \in SV} \alpha_i K(x_i, x)$$

where SV is the set of support vectors. Observe that the model is sparse with respect to α. The most well known kernels (from which one can derive matrix K) are Linear, Polynomial and Gaussian. A small word must be said on the different parameters that will be trained in order to favor interpretability of the model. C penalizes misclassifications. The larger the C the smaller the distance to the margin will be between two classes and the model may not generalize well. In the radial basis kernel where ϕ is gaussian (the feature space has inf dimension), the γ parameter controls the shape of the decision boundary. Note that $\gamma = \frac{1}{2\sigma^2}$. In a Gaussian distribution, a high σ^2 is a flatter curve (more variance). This translates to a small γ. In the Gaussian RBF, a low value for γ the curvature of the decision boundary is lower. A large value for γ may create a decision boundary adapted to training set and return an overfitted model. d is the degree of the polynomial kernel. Observe that finding $\phi(x)$ explicitly shows that the dimension of the feature space is $\binom{n+d}{d}$. c in the polynomial kernel acts as a trade-off of the influence of low dimension terms vs. the high ones. If $c = 0$ the polynomial kernel is called homogeneous kernel. Finally, the assumption that the linear, polynomial and rbf kernels are actually kernels have been made. Proofs can be found in Proposition 3.24 of [24].

Table 5.4 shows the validation scores for different combinations. A grid search (without cross validation, it was deemed unnecessary because of the data set size) has been implemented to find the optimal C (and γ and d, in the case of the RBF and Polynomial kernel). The models are trained on parameters are $C \in \{10^{-4}, 10^{-3}, 0.01, 0.1, 1, 10, 100\}$, $\gamma \in \{10^{-5}, 10^{-4}, 10^{-3}, 0.01, 0.1, 1\}$ and $c \in \{0, 1, 10\}$, $d \in \{2, 3, 4\}$.

Feature vector	Model w. best parameters	Weighted avg	Micro F1
Xception	Linear, $C = 0.01$	53.08%	53.40%
	RBF, $C = 10$, $\gamma = 0.01$	56.05 %	57.05%
	Poly, $C = 1$, $c = 0, d = 3$	56.66 %	57.30%
EfficientNetB7	Linear, $C = 1$	31.94%	36.64%
VGG19	Linear, $C = 0.1$	50.70%	51.00 %
	RBF, $C = 10$, $\gamma = 0.1$	52.72%	53.23%
	Poly, $C = 0.1, c = 10, d = 3$	50.04%	48.96 %
Inception_Resnet_v2	Linear, $C = 0.01$	54.30 %	54.65 %
	RBF, $C = 10$, $\gamma = 0.01$	55.52 %	56.32%
	Poly, $C = 10c = 0, d = 2$	54.51%	54.61%
SIFT	Linear, $C = 0.1$	35.12 %	36.64 %
Xception-EfficientNetB7 -VGG19-Inception_Resnet -SIFT	Linear, $C = 0.01$	58.79 %	59.17 %
Xception -VGG19 -Inception_Resnet -SIFT	Linear, $C = 0.01$	58.98%	59.17%
	RBF, $C = 10, \gamma = 0.01$	58.55%	60.22 %
	Poly $C = 10, c = 0, d = 2$	59.56%	59.73 %

Table 5.4: Results for SVM with different feature vectors

These values are chosen based on previous results and theoretical grounds. Observe that in some cases a smaller grid may be needed due to time constraints and computational costs.

Xception, VGG19 and Inception_Resnet_v2 are the clear winners as feature vectors. The RBF and Polynomial kernel clearly outperform the Linear in all instances, however, between them it depends on which feature they are applied. Observe that the Linear tends to find its highes weighted average on a lower C, the opposite of the RBF and Polynomial kernel.

5.1.1.2 Non-standard kernels. Multiple Kernel Learning (MKL)

Once the standard kernels (linear, rbf, polynomial) have been put to test, it is reasonable to explore further options. The field of kernel design shows that new kernels can be formed via matrix operations.

When working with multiple kernels, one must distinguish two different cases. First, the sum or product of two kernels over the same space X. Second, the sum or product of two kernels belonging to two different spaces, i.e k_1 over X_1 and k_2 over X_2. The resulting kernel k is over the cartesian product space $X = X_1 \times X_2$. In practical usage and specifically in this report, the sum over the same space is used to test different kernels, where as the sum over several spaces is to test the importance of the different feature vectors.

The proposition below formalizes the aformentioned, since the positive semidefiniteness of these new kernels must be guaranteed.

Proposition 1. The following functions are kernels:

(i) Let X be a vector space. Given k_1, k_2 over X. Then, for $x^i, x^j \in X$, $k(x^i, x^j) = k_1(x^i, x^j) + k_2(x^i, x^j)$ is a kernel over X. (sum of kernels)

(ii) Under the hypothesis of (i), $k(x^i, x^j) = k_1(x^i, x^j) \cdot k_2(x^1, x^j)$ where $\cdot$ denotes their element-wise product. (product kernel)

(iii) Let X_1, X_2 be vector spaces with different dimension. Given k_1 kernel over X_1 and k_2 over X_2 and $x^i, x^j \in X_1$ and $x^{i'}, x^{j'} \in X_2$, the direct sum (defined over $X = X_1 \times X_2$) $k = k_1 \oplus k_2$ is computed as: $k\big((x^i, x^{i'}), (x^j, x^{j'})\big) = k_1(x^i, x^j) + k_2(x^{i'}, x^{j'})$.

(iv) Under the hypothesis of (iii), the tensor product $k = k_1 \otimes k_2$ (defined over $X = X_1 \times X_2$) is computed as:
$k\big((x^i, x^{i'}), (x^j, x^{j'})\big) = k_1(x^i, x^j) \cdot k_2(x^{i'}, x^{j'})$ ($\cdot$ also refers to the element wise product).

(v) Given $a > 0$ and a kernel k over X, $a \cdot k$ is a kernel.

Proof. (i), (ii) and (v) can be found in [24]. Also, when studying the feature spaces below, a partial proof for (i) and (ii) is given. For (iii) and (iv), a possible proof is to consider k_1 psd kernel and $\phi_1 : X_1 \to \mathcal{H}$ such that:

$$k_1(x^i, x^j) = \langle \phi_1(x^i), \phi_1(x^j) \rangle_{\mathcal{H}}$$

Let $\phi : X_1 \times X_2 \to \mathcal{H}$ defined by the projection of the first component:

$$\phi((x^i, x^{i'})) = \phi_1(x^i)$$

For $x_1 = (x^i, x^{i'})$ and for $x_2 = (x^j, x^{j'})$ from X:

$$\langle \phi((x^i, x^{i'})), \phi((x^j, x^{j'})) \rangle_{\mathcal{H}} = k_1(x^i, x^{i'})$$

which is by hypothesis a positive semidefinite kernel over $X_1 \times X_2$. Analogously k_2 is positive semidefinite. Now, define the direct sum and the tensor product which are positive semidefinite by (i) and (ii) of Proposition 1.

□

The formal framework for the items in Proposition 1 is Multiple Kernel learning (MKL). In [25] there is an extensive survey on the theory sustaining Multiple Kernel Learning (MKL). The following paragraphs are to introduce MKL and to set some notations (following [25]). The main idea is to consider several kernels $k_1, \ldots, k_M$ and a combination function $f : \mathbb{R}^M \to \mathbb{R}$, such that, for data points x^i, x^j

$$k_{cmb} = f_\mu\big(\{k_m(x_m^i, x_m^j)\}_{m=1}^M\big)$$

where $X_m^i \in \mathbb{R}^{d_m}$, $k_m : \mathbb{R}^{d_m} \times \mathbb{R}^{d_m} \to \mathbb{R}$ $\forall m = 1, \ldots, M$ (d_m is the dimensionality of the feature space, see table 5.1 for details). Note that f may be non-linear and that the parameters μ define the "importance" of one of the kernels. There is also the possibility of μ being part of the kernel function, but these methods are less common. Hence, the study restricted to the case where μ parameters are optimized during training.

There are different ways of learning an MKL based model (fixed rules, heuristically...). In this report, an optimization problem is posed with the goal of finding the best combination of μ that maximize the margin.

For instance, if one considers the weighted sum kernel, the regularization problem to solve in MKL is then:

$$\min_{f_1, \ldots f_M} \left\{ \frac{1}{n} L\Big(\sum_{i=1}^M f_i(x^i), y^i \Big) + \lambda \sum_{i=1}^M \frac{\|f\|^2_{\mathcal{H}_{K_i}}}{\mu_i} \right\} \tag{5.2}$$

where L is the hinge loss. The following theorem guarantees a solution f^* to problem 5.2 via the Representer Theorem.

Theorem 1. Problem 5.2, when learning with the weighted sum kernel $k_\mu(x, x') = \sum_{i=1}^M \mu_i k_i(x, x')$ with $\mu_1, \ldots \mu_M \geq 0$ admits a solution f^* defined as:

$$f^* = \sum_{i=1}^M f_i^*$$

where $(f_1^*, \ldots, f_M^*) \in \mathcal{H}_{K_1} \times \ldots \times \mathcal{H}_{K_M}$.

Proof. Can be found in [26]. □

Moving from the theoretical to the practical field, from table 5.4, it is clear that on their own, Xception and Inception_Resnet_v2 return the best results on the Linear, RBF and Polynomial kernels. One can consider the weighted sums:

$$k = \beta k_{linear} + (1 - \beta) k_{RBF}$$
$$k = \beta k_{RBF} + (1 - \beta) k_{Poly}$$

It is clear by Proposition 1 ((i) + (v)) that k is a kernel function. Now β is a new parameter to consider in the parameter grid. Observe that $\beta \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ Similarly to the previous section,For the RBF and Polynomial kernel, models are trained on the following hyperparameters $C \in \{0.01, 0.1, 1, 10\}$, $\gamma \in \{10^{-3}, 0.005, 0.01, 0.05, 0.1\}$ and $c \in \{0, 1\}$, $d \in \{2, 3, 4\}$. Table 5.5 shows small improvements over the baseline results.

Feature vector	Best weighted Kernel	Weighted avg	Micro F1.
Xception	$C = 1$, 0.3Lin() + 0.7RBF($\gamma = 0.1$)	53.70%	54.00%
Xception	$C = 10$, 0.1RBF($\gamma = 0.1$) + 0.9Poly($d = 4$, $c = 1$)	55.53%	56.81%
Inception_Resnet_v2	$C = 10$, 0.4Lin() + 0.6RBF($\gamma = 0.1$)	54.23%	54.53%
Inception_Resnet_v2	$C = 10$, 0.1RBF($\gamma = 0.1$) + 0.9Poly($d = 4, c = 1$)	54.99%	56.12%
VGG19	$C = 10$, 0.1Lin() + 0.9RBF($\gamma = 0.05$)	51.69%	51.36%
VGG19	$C = 10$, 0.9RBF($\gamma = 0.05$) + 0.1Pol($d = 2$, $c = 0$)	53.28%	53.36%

Table 5.5: Weighted sum of kernels, best combinations of parameters

The weighted sum of kernels finds the adequate β_i parameters, which can be interpreted as the importance of each kernel. The sum of the two kernels in the feature space is the concatenation of vectors $\phi_1(x), \phi_2(x)$, yielding $\phi(x) = [\phi_1(x), \phi_2(x)]$. The proof of of this is:

$$k(x^i, x^j) = k_1(x^i, x^j) + k_2(x^i, x^j) = \langle \phi_1(x^i), \phi_1(x^j) \rangle + \langle \phi_2(x^i), \phi_2(x^j) \rangle = \langle \phi(x^i), \phi(x^j) \rangle$$

The last equality follows from noting that commutative property of the sum of real numbers. The dimension of the feature map is clearly the dimension of the concatenated vector.

The choice of summing these different kernels is to see how they compensate each other. Given a linear kernel, it can be interesting to see how adding gaussian features can construct a kernel that can create a better separating hyperplane. The same applies for the combination of Polynomial and RBF: Recall that the polynomial has a term c gives weight to lower order terms.

Recall that in table 5.4 the concatenation of the different feature extraction vectors was considered, and gave significantly good results. Now it is clear that this is guaranteed by the summation kernel.

For the three feature vectors shown in the table, the RBF kernel on its own still performs better than any combination. However, the combination of the Polynomial + RBF improves significantly the weighted average score of VGG19 (by 3%) and Incpetion_Resnet_v2 (0.5 %). The Linear + RBF sum is better in all three vectors compared to the vectors trained on Linear Kernel, as it allows for Gaussian features to be part of the decision boundary.

By (ii), product kernels can also be considered. It is important to understand what the feature space is. It is clear from (ii) that there is a product between all pairs of components of ϕ_1 and ϕ_2:

$$\begin{aligned} k(x^i, x^j) &= k_1(x^i, x^j) k_2(x^i, x^j) = \langle \phi_1(x^i), \phi_1(x^j) \rangle \cdot \langle \phi_2(x^i), \phi_2(x^j) \rangle = \\ &= \left(\sum_{k=1}^{D_1} \phi_1(x^i)_k \phi_1(x^j)_k \right) \left(\sum_{l=1}^{D_2} \phi_2(x^i)_l \phi_2(x^j)_l \right) \\ &= \sum_{k=1}^{D_1} \sum_{l=1}^{D_2} \phi_1(x^i)_k \phi_1(x^j)_k \phi_2(x^i)_l \phi_2(x^j)_l = \sum_{k=1}^{D_1} \sum_{l=1}^{D_2} \big(\phi_1(x^i)_k \phi_2(x^i)_l\big) \big(\phi_1(x^j)_k \phi_2(x^j)_l\big) \end{aligned}$$

Observe that one can define the feature map ϕ as:

$$\phi(x^i) = \begin{bmatrix} \phi_1(x^i)_1 \phi_2(x^i)_1 & \phi_1(x^i)_1 \phi_2(x^i)_2 & \cdots & \phi_1(x^i)_{D_1} \phi_2(x^i)_{D_2} \end{bmatrix}^T$$

To finally obtain $k(x^i, x^j) = \langle \phi(x^i), \phi(x^j) \rangle$. So, from the point of view of the product kernel, it can be interesting to consider higher order interactions between the images of ϕ_1 and ϕ_2. Since $\phi(x)$ is defined as component wise products of ϕ_1, ϕ_2, a the points may be mapped into a feature space where finding the separating hyperplane is easier.

Table 5.6 shows the results. The product of RBF kernels seems to have little effect on performance (increases up to 0.5% are observed). Products between Linear and RBF kernel always perform better

Feature vector	Best Product Kernel	Micro F1	Weighted avg
Xception	$C = 10$, Linear $\cdot RBF(\gamma = 0.05)$	53.23 %	56.00%
Xception	$C = 10$, Poly$(d = 4,\ c = 1)$RBF$(\gamma = 0.05)$	54.02%	56.56%
Xception	$C = 0.01$, RBF$(\gamma = 0.01)$RBF$(\gamma = 0.01)$	56.57%	57.78%
Inception_Resnet_v2	$C = 10$, Linear$\cdot$RBF$(\gamma = 0.01)$	54.58%	53.78%
Inception_Resnet_v2	$C = 10$, Poly$(d = 4,\ c = 1)$RBF$(\gamma = 0.01)$	55.14%	54.35%
Inception_Resnet_v2	$C = 10$, RBF$(\gamma = 0.01)$RBF$(\gamma = 0.01)$	55.80%	54.43 %
VGG19	$C = 0.1$, Linear $\cdot$ RBF$(\gamma = 0.05)$	52.64%	52.46%
VGG19	$C = 10$, Poly$(d = 4,\ c = 1, d = 1)$RBF$(\gamma = 0.05)$	52.63%	52.87%
VGG19	$C = 10$, RBF$(\gamma = 0.05)$RBF$(\gamma = 0.05)$	52.72 %	53.23 %

Table 5.6: Product of kernels, best combinations of parameters

than the Linear kernel, but do not manage to outperform the RBF kernel. Finally, the product of the RBF with Polynomial gives significantly better results compared to a single Polynomial kernel (except in the Xception veector).

For more general MKL practice, in Python the `MKLpy` package (developed by Ivano Lauriola at Università di Padova) [27] allows for the implementation of MKL learning. One of the strengths of this package is that it has functions for the whole pipeline (preprocessing, kernel computation, SVM fitting and evaluation). It has support for many functions as well as many MKL algorithms. It is not the purpose of this report to delve into the different kinds of MKL algorithms. Without going deep into their technicalities, the following are tried: AverageMKL, EasyMKL, PWMK.

1. **AverageMKL.** As its name suggests, kernel k_μ is defined as $k_\mu = \sum_{i=1}^{M} \mu_i k_i(x, x')$ with $\mu_i = \frac{1}{M}$

2. **EasyMKL.** (Proposed by Fabio Aiolli and Michele Donini, [28]).
   In this case, $k_\mu = \sum_{i=1}^{M} \mu_i k_i(x, x')$ with $\mu_i = \frac{1}{M}$ finds the best μ_i convex combination, with $\mu_i \geq 0$ and $\|\mu\|_1 = 1$. Also note that there is a λ hyperparameter called `lam` (not to be confused with the $\lambda \in [0, 1]$ in the formulation of 5.2). With $\lambda = 0$, the algorithm focuses on maximizing margin between classes, and with $\lambda = 1$, the algorithm maximizes distance between centroids. Since SVM is a maximum margin classifier, models will be trained with $\lambda = 0$.

3. **PWMK.** It takes a Heuristic approach and finds the μ_i weights of kernel k_μ taking into account individual kernel performance.

$$\mu_r = \frac{acc(Kr) - m\delta}{\sum_h \big(acc(K_h) - m\delta\big)}$$

   where m is the minimum accuracy and δ is a hyperparameter. The Python method accepts a `cv` parameter, which is used to compute the accuracy. For clarity, $\delta = 0$ is taken.

Feature vector	Algorithm	Micro	Weighted
Xception, VGG19, Inception_Resnet, SIFT	AverageMKL, $C = 10$	59.90%	58.70 %
Xception, VGG19, Inception_Resnet, SIFT	EasyMKL, $C = 10$	60.23 %	59.06 %

Table 5.7: Multiple Kernel Algorithms with RBF Kernel results

Something to note about the output of `EasyMKL` is that it returns the parameters μ_i it learned. For the model trained, EasyMKL gives more importance to the first kernel it gets, by assigning it a larger weight. This has an impact on the weighted score, which changes according to the order of kernels. The results in this report show the maximum score achieved on trying different combinations.

Table 5.7 shows the results in the MKL context. The **direct sum** of the 4 kernels (4 RBF kernels are computed for each feature vector). Observe that the weighted score is high, but still performs lower than the score obtained on the concatenated vector.

5.1.1.3 Deep kernel learning. Multiple-Layer Multiple Kernel Learning (MLMKL)

Papers [29] and [30] give an extensive introduction to the concept of Deep kernel learning. In some specific problems, MKL has shown to fall short, especially if the k_{comb} is only a convex combination of P kernels, because it is not capable of reaching NN accuracy. Deep Kernel Learning is the learning architecture that consists of a concatenation of a kernel function with non-linear functions, as well as kernel functions. An $l - layer$ kernel is defined as the composition of a kernel l times:

$$k^{(l)}(x^i, x^j) = \langle \phi^l(x_i), \phi^l(x_j) \rangle = \langle \phi(\phi(\cdots \phi(x^i))), \phi(\phi(\cdots \phi(x^j))) \rangle$$

Unfortunately, a lack of time (both in the theoretical and programming field did not allow to try some simple combinations). However, it must be noted that a Representer Theorem has been formulated for Deep Kernel Learning has been formulated both for finite and infinite data samples.. Completely out of reach of this report, references are found in [31].

5.1.2 Multilayer Perceptron

Instead of using SVMs on one or more previously extracted feature vectors, feed-forward densely connected Neural Networks, called Multilayer Perceptrons (MLP from now on), can be used as well. All code covering the following discussion can be found in `3_2_MLP_training.ipynb`.

5.1.2.1 Loss Function

Three loss function are considered for the problem at hand:

1. The Cross-entropy Loss (CEL from now on). This is the most commonly used loss function for multi-class classification tasks in Neural Network training and is defined as:

$$\text{CEL} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} 1_{y_i \in C_c} \log p_{\text{model}} \ [y_i \in C_c], \tag{5.3}$$

   with N the number of observations, C the number of classes, and $y_i \in C_c$ signifying the i'th observation belonging to the c'th class. Note however that weight updating using this loss function can be prone to being dominated by the majority classes because more examples of these classes are given, hence more strongly influencing gradient descent. Subsequently, the model might optimize overall accuracy at the possible cost of poor minority class prediction.

2. The Weighted Cross-entropy Loss (WCEL from now on) is a first possible solution to the previously stated imbalance problem. It extends CEL by penalizing misclassification of minority classes by in CEL replacing the one-hot encoded y_i by class weights α_c resulting in definition

$$\text{WCEL} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} \alpha_c \log p_{\text{model}} \ [y_i \in C_c], \tag{5.4}$$

   where α_c can be defined in multiple ways but is generally proportional to the inverse of the class proportion. In this case, it is defined as $\alpha_c = \frac{N}{C * bincount(c)}$, where bincount(c) is the amount of images in class c.

3. The Focal Loss (FL from now on) is another candidate solution to the imbalanced classification bias possibly introduced by using CEL. It introduces a 'focussing' parameter γ that helps distinguishing between hard-to-predict and easy-to-predict cases by reducing the influence on the loss function of correct and highly confident predictions. The loss function is defined as

$$\text{FL} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} \alpha_c (1 - p_{\text{model}} [y_i \in C_c])^\gamma \log p_{\text{model}} \ [y_i \in C_c], \tag{5.5}$$

   where $\gamma = 2$ is the standard value used often obtaining good performance [32].

5.1.2.2 Parameter Optimization

For each loss function, a grid is created that outputs model configurations of up to three layers with different possible sizes ((512, 256 or 128) for layer 1, (256 or 128) for layer 2 and (128) for layer 3). Two optimizers are considered: Adaptive Moment Estimation (Adam) and Stochastic Gradient Descent (SGD). The learning rates included in the grid are 0.001, 0.005, 0.01 and 0.05.

5.1.2.3 Regularization

In order to avoid overfitting, two regularization mechanisms are implemented:

1. Early Stopping: This implies keeping track of performance on train- and validation set in order to pinpoint from which point training loss increasing while validation doesn't (overfitting). Therefore, the point of minimum validation loss is considered the point after which overfitting takes place. Consider the best model the one at the point of this change. It is always made sure enough epochs are ran for each model that this loss minimum is most probably reached during training. An example of the training- and validation curves can be found in Figure 5.1, which comes from the training of the model with the highest weighted average F1-score. This is the one using as input the concatenation of all features and the ordinary CEL for training.

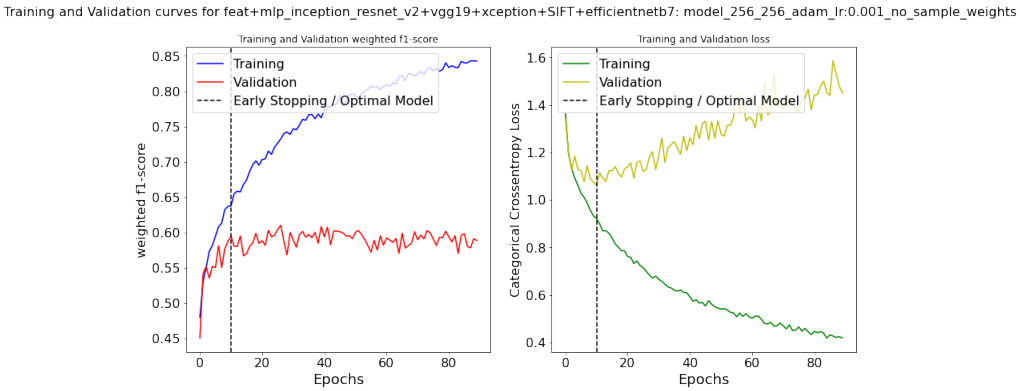

Figure 5.1: Training and validation curves with early stopping of the model with highest weighted average f1-score.

2. Dropout: This is a layer that in this case is added between the input vector and the first neuron layer, although it can be implemented between any layer in the model. It randomly sets a proportion of the input to zero, forcing the model to not be able to see all neurons/features at every epoch. This works as a regularizer as it prevents overly relying on certain features and/or neurons (overfitting) by reproaching its availability hence promoting equal importance across features/neurons.

5.1.2.4 Results

The training results are presented in Table 5.8. A few things can be concluded:

- The best performing single feature is Xception (with WCEL) such that this pretrained network will be used as the convolutional base in the fine-tuning model in section 5.2.
- As expected, the models with concatenated vectors perform better than the single vector models. The MLP model with the highest weighted average F1-score uses all features available. This shows that in future work, adding new feature vectors might even further improve performance.
- The best performing model is trained using the non-weighted CEL, proving that weighting the classes or using focal loss does not per se improve performance. However, note that the results for different loss functions for each method lie close to each other. So given that

Table 5.8: Results of parameter optimization and model validation for models combining feature extraction and a Multilayer Perceptron (MLP).

Feature Vector	**loss fn.**	**Optimal Parameters**	**validation set weighted avg F1-score**	**validation set accuracy**
Xception	CEL	512, SGD, lr:0.01	54.27 %	56.73 %
	WCEL	512, SGD, lr:0.01	55.61 %	56.73 %
	FL	512, SGD, lr:0.01	55.19 %	57.38 %
EfficientNetB7	CEL	512, SGD, lr:0.01	24.98 %	30.99 %
	WCEL	512, SGD, lr:0.01	24.01 %	28.02 %
	FL	512, SGD, lr:0.01	23.47 %	30.42 %
VGG19	CEL	256-256, adam, lr:0.001	49.96 %	53.03 %
	WCEL	256-256, adam, lr:0.001	47.78 %	48.72 %
	FL	256-256, adam, lr:0.001	49.72 %	52.54 %
Inception_Resnet_v2	CEL	512, SGD, lr:0.01	53.94 %	56.28 %
	WCEL	512, SGD, lr:0.01	53.83 %	55.10 %
	FL	512, SGD, lr:0.01	52.00 %	54.74 %
SIFT	CEL	512-256, SGD, lr:0.05	38.18 %	42.82 %
	WCEL	512, SGD, lr:0.01	39.68 %	40.18 %
	FL	512, SGD, lr:0.01	37.96 %	43.23 %
Xception-EfficientNetB7 -VGG19-Inception_Resnet -SIFT	**CEL**	**256-256, adam, lr:0.001**	**59.49 %**	60.88 %
	WCEL	512, SGD, lr:0.01	58.42 %	59.33 %
	FL	512, SGD, lr:0.01	56.99 %	58.36 %
Xception-Inception_Resnet -VGG19-SIFT	CEL	256-256, adam, lr:0.001	58.90 %	61.28 %
	WCEL	256-256, adam, lr:0.001	59.22 %	58.89 %
	FL	256-256, adam, lr:0.001	58.06 %	60.51 %
Xception-Inception_Resnet -VGG19	CEL	512, SGD, lr:0.01	56.09 %	58.97 %
	WCEL	256, SGD, lr:0.01	57.27 %	59.05 %
	FL	512, SGD, lr:0.01	56.94 %	58.80 %

there is a random component to Neural Network model training, in another iteration, a model trained using WCEL might just as well prove to have the best performing model. Note that FL is typically used for extremely imbalanced training, more than present in this dataset. This might explain that it does not increase model performance. To see if the WCEL and CEL deliver different models, its validation set confusion matrices for the model using all feature vectors are compared in Figure 5.2. From these it becomes clear both models perform similarly on most classes. However, the model using WCEL focusses slightly more to predict the two apparently most difficult classes (Neoclassicism and Renaissance) than the CEL model, resulting in decreased accuracy for some of the more occurring classes hence leading to a lower weighted average F1. This more balanced aproach is what is expected from a WCEL model. The differences are however small and the two models can be considered as performing similarly.

- Generally speaking pre-trained network feature vectors outperform SIFT, except for EfficientNetB7 which performs only slightly better than a random guessing model.

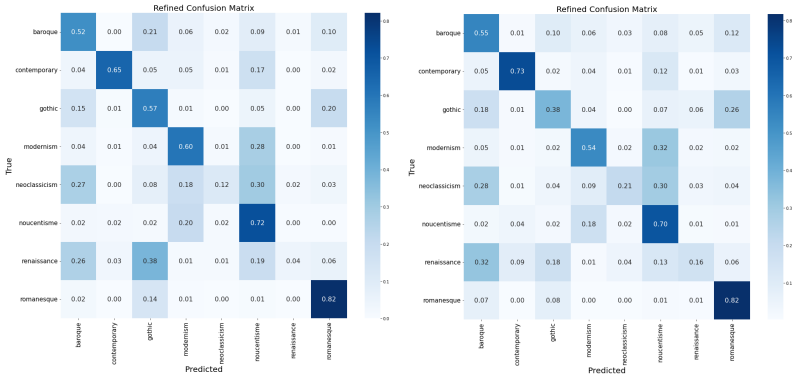

Figure 5.2: Normalized validation set confusion matrices for optimal MLP model using concatenation of all feature vectors using CEL (left) and WCEL (right).

5.2 Convolutional Neural Network (CNN) Based Methods

5.2.1 Fine-Tuning a Pre-Trained CNN

Just like in feature extraction, this fine-tuning technique considers using a pre-trained network, again stripping it from its top layer. However, in this case, one or more layers are directly attached to the model, compared to extracting features and building an MLP seperately. When the convolutional (pre-trained) layers are then frozen, the model is equivalent to the feature extraction - MLP model. Therefore, for any choice of pre-trained network, the corresponding optimal architecture obtained in Section 5.1.2 can be chosen as top layer, therefore requiring less parameter optimization. Up to this point, the only difference between seperate feature extraction and transfer learning is that random image augmentation is used. It's considered a good practice to artificially introduce sample diversity by applying random transformations to the training images, such as flipping, rotating and zooming [33]. This helps expose the model to different aspects of the training data while slowing down overfitting as well as oversampling the minority classes that might otherwise be underrepresented.

The following and most important step in transfer learning is to unfreeze one, many or all convolutional blocks of the pre-trained network and to train the weights on a low learning rate, typically an order of magnitude lower than what is used for top layer training. This lower learning rate is chosen as the convolutional layers typically represent more general image features like edges and trivial geometry that generalize well and need only slight adaptations for potentially meaningful results [33]. Note that updating the weights of big pre-trained models is computationally expensive. A rented GPU is a minimal requirement that still implies hours if not days of training. For this reason, only a single transfer learning model is build.

The application of the theory mentioned above on the problem at hand can be found in notebook `3_3_TF_training.ipynb`. Because Xception is the best performing extracted feature of the models in Section 5, it is chosen as convolutional base for fine-tuning. Similarly, the same parameter setup for the top layer is used that is found in the optimal Xception model in Section 5.1.2: A single layer of 512 units, the SGD optimizer with a learning rate of 0.01 and the weighted cross-entropy loss. Just like in the MLP models, in the first round of training considering only the top layer, the model with the lowest WCEL is chosen, which results in a model with weighted average F1-score of 54.31 %. In the second round all weights are unfrozen and the model trained with a learning rate of 0.001. The best model, by selecting the lowest loss, obtains a weighted average F1-score of 71.97 %.

5.3 Model Comparison

For each of the three methods tried, the best models performance on the validation is compared in order to select a final model. An overview of the three candidate models in shown in Table 5.9, a classification report of each of the models is presented in Figure 5.3 and the respective normalized confusion matrices can be found in Figure 5.3.

From these results it becomes clear that the transfer learning model shows the lowest generalization error with a weighted average F1-score of 71.71 % on the validation set. Furthermore, the confusion matrices clearly show that it classifies minority classes significantly more accurately then the feature extraction based models. It can thus be concluded that transfer learning is the most fitting method of the ones covered in this study. It makes efficient use of large pre-trained networks but seems to significantly outperform statically extracted features by slightly changing the weights of the entire network in the direction of the task at hand.

Table 5.9: Overview of the best performing models of the three major methods tried

Method	**Parameters**	**Weighted Average F1-score**
Feature Extraction + SVM	Feature Vectors: Xception-VGG19-Inception_Resnet-SIFT $C = 0.01$	59.56 %
Feature Extraction + MLP	Feature Vectors: Xception-EfficientNetB7-VGG19-Inception_Resnet-SIFT Loss Function: CEL Layers: 256-256, Optimizer: adam Learning Rate: 0.001	59.49 %
Transfer Learning	Convolutional Base: Xception Top Layer: 512, Otimizer: SGD Learning Rate (Top Layer): 0.01 Learning Rate (All Weights): 0.001	**71.97 %**

	precision	recall	f1-score	support
baroque	0.47	0.58	0.52	305
contemporary	0.64	0.62	0.63	178
gothic	0.46	0.47	0.47	214
modernism	0.63	0.57	0.60	499
neoclassicism	0.33	0.33	0.33	186
noucentisme	0.63	0.63	0.63	529
renaissance	0.29	0.15	0.20	68
romanesque	0.81	0.82	0.81	480
accuracy			0.60	2459
macro avg	0.53	0.52	0.52	2459
weighted avg	0.60	0.60	0.60	2459

	precision	recall	f1-score	support
baroque	0.46	0.55	0.50	305
contemporary	0.75	0.73	0.74	178
gothic	0.42	0.38	0.40	214
modernism	0.64	0.54	0.58	499
neoclassicism	0.52	0.21	0.30	186
noucentisme	0.56	0.70	0.62	529
renaissance	0.17	0.16	0.17	68
romanesque	0.76	0.82	0.78	480
accuracy			0.59	2459
macro avg	0.54	0.51	0.51	2459
weighted avg	0.59	0.59	0.58	2459

	precision	recall	f1-score	support
baroque	0.65	0.70	0.68	289
contemporary	0.72	0.95	0.82	176
gothic	0.63	0.66	0.65	214
modernism	0.75	0.66	0.70	468
neoclassicism	0.50	0.67	0.57	177
noucentisme	0.77	0.63	0.69	517
renaissance	0.56	0.93	0.70	76
romanesque	0.90	0.79	0.84	496
accuracy			0.72	2413
macro avg	0.68	0.75	0.71	2413
weighted avg	0.73	0.72	0.72	2413

Figure 5.3: Comparison of validation set classification reports for best performing model of the three major methods tried: feature-extraction + SVM (left), feature-extraction + MLP (middle) and transfer-learning (right)

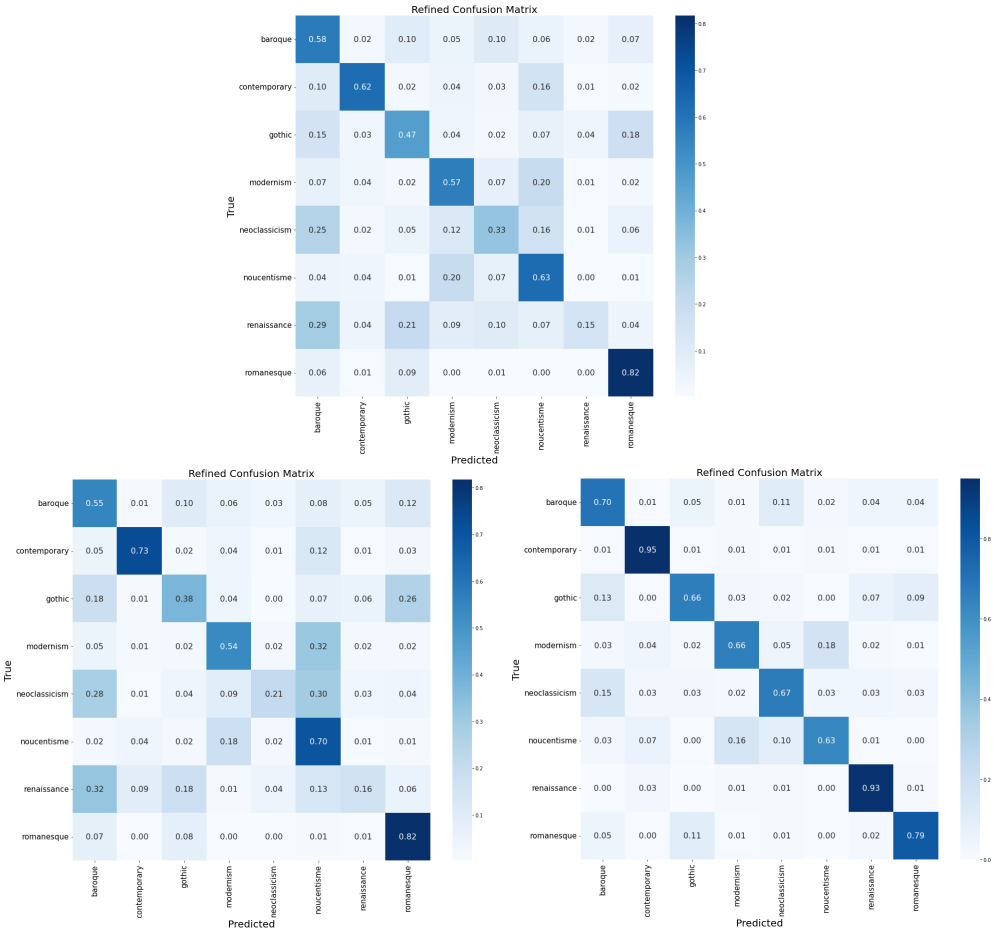

Figure 5.4: Comparison of validation set confusion matrices for best performing model of the three major methods tried: feature-extraction + SVM (Top), feature-extraction + MLP (bottom-left) and transfer-learning (bottom-right).

6 Final Model

As explained in Section 5.3, the Xception-based transfer learning model described in Secion 5.2.1 is considered the best performing model. For a final estimation of its generalization error, the model is evaluated on the test set. Note that it is considered a good practice to retrain the model on the training and validation set to make use of all images available. However, in this case, training is too expensive and computing resources scarce, that it is chosen to not re-train.

In Figures 6.1 and 6.2, the final test set classification report and confusion are respectively presented. The following conclusions can be made from the results:

- Model validation underestimated the generalization error as the weighted F1-score drops from 71.97 % to 57.94 % on the test set. A possible explanation for this is the fact that the data is highly varied such that validation- and test-set might have been of varying difficulty. A second explanation is overfitting on the minority classes. As oversampling is used during transfer learning training, the few minority class images are reused using random transformations. Although this process is not applied on the validation set, the model is chosen at the lowest validation loss, which might be a point at which the few images in the validation set happen to be classified correctly based on the oversampled training images. However, this is prone to not generalizing perfectly, as can clearly be seen on the confusion matrix where minority classes like Renaissance and Neoclassicism respectively drop from 93 % and 67 % accuracy on the validation set to 42 % and 35 % on the test st, a difference strongly impacting the F1-scores.

- For each of the true classes, the most commonly predicted class is the correct one. This means that no class is disregarded in favour of majority classes, probably due to the weighting in WCEL in training and using Weighted average f1-score instead of overall accuracy. The model can hence be considered balanced.
- The four classes that not reach 50% precision are 'Baroque', 'Gothic', 'Neoclassicism' and 'Renaissance'. Although these have in common that they can be considered minority class, it cannot be the only reason, because 'Contemporary', another minority class, performs strongly relative to the other classes with 82% precision. Therefore, the nature of these categories seems to influence the difficulties in classification. An example of this is that the styles overlap in time period. For example, Gothic is most often classified as Romanesque, which overlap in Catalunya in the 13th century [1]. It seems also to be the reason that contemporary styles are classified so well. On the other hand, some styles simply share more characteristics than others. An example of this is Neoclassicism being described as outgrowing from the Late Baroque period and Noucentisme frequently implementing Classicist foundations , which for neoclassicism are the two most often made misclassifications [1].
- Holding into account the noisiness and complexity of the data described in Section 3.4, the model can be considered performing strongly. In fact, it is dared to hypothesize that the model is not too far from an upper bound if only the current image set is held into consideration.

	precision	recall	f1-score	support
baroque	0.55	0.49	0.52	409
contemporary	0.60	0.82	0.69	221
gothic	0.40	0.47	0.43	264
modernism	0.66	0.51	0.57	628
neoclassicism	0.27	0.42	0.33	224
noucentisme	0.63	0.56	0.59	695
renaissance	0.20	0.35	0.26	85
romanesque	0.81	0.74	0.78	609
accuracy			0.57	3135
macro avg	0.51	0.54	0.52	3135
weighted avg	0.60	0.57	0.58	3135

Figure 6.1: Classification Report of the Xception-based fine-tuning model on the test set.

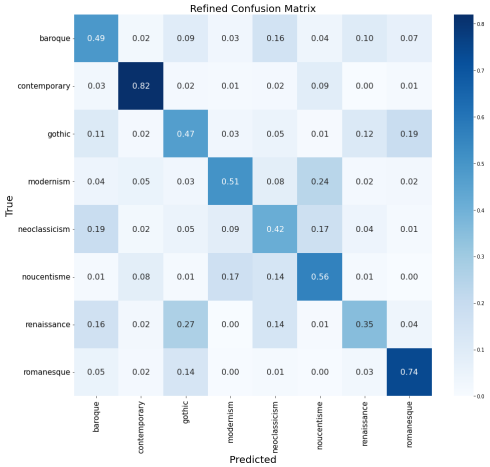

Figure 6.2: Confusion Matrix of the Xception-based fine-tuning model on the test set.

7 Conclusions

In this study, an image classification model is developed to differentiate between eight prominent styles of Catalan architecture. Both deep learning and non-deep learning methods are explored.

The non-deep learning methods make use of feature extracted vectors. These are obtained by

using both pre-trained CNN features and non pre-trained network feature extractors like SIFT. The features from pre-trained CNNs generally outperform the SIFT-one. Now using these features, two major classical machine learning methods are applied: SVMs using kernel methods and MLPs.

Kernel design is broad field of research. Once mathematical properties are verified, the possibilities of constructing new kernels are virtually unlimited. In this project the success was to improve in many cases the baseline accuracy using new kernels obtained via the closure properties. However, it must be said that the best result was obtained via an MKL approach but rather by concatenating the vectors into a single one. This proves that for this specific task, a single kernel works better for a longer vector that than considering more than a sum of each kernel per feature vector although the result is close in terms of the validation weighted score . Nonetheless, one should note the limitations of the MKL, only three types of kernels were considered, and sums and products were restricted to. As stated, the field and literature in it is vast.

MLP training is another natural choice to apply on the feature vectors. Its often non-linear nature and flexibility in choice of neuron architecture, loss function and regularization allows for powerful use of concatenated input vectors. It is found that the best MLP model is found to perform similarly to the optimal SVM model.

The deep-learning approach considered is transfer learning, which encompasses extending the architecture and updating the weights of a pre-trained network. The approach using Xception as convolutional base and a single top layer yields the best validation results of all models in the study. On the test set, an average weighted F1-score of 57.94 % has been reached.

While the best model predicts the right label for each image and each class more often than not, it shows limitations in classifying some of the categories. This can be attributed to a combination of factors, including the quality of the data, the complexity of the problem, and high computational demand. Despite these limitations, the results demonstrate the potential of using both deep learning and non-deep learning methods for image classification tasks in the architectural field.

8 Limitations and Outlook

One major limitation throughout the project has been computational cost. Repeated Paid Google Colab Subscriptions where necessary for model training. Therefore, in a future iteration of the project, with more computational power and time, the following ideas could be realized:

- Trying more transfer learning model architectures.
- Trying more feature extractors.
- Using bigger parameter grids.

A second limitation is the noisiness and complexity of the image set. A more manually curated approach including multiple source integration could severely improve results.

For all methods using feature extraction, given that performance increased by adding more input vectors, an approach could be to add even more pre-trained network vectors.

The limitations to SVM and kernel methods are clear: Firstly, computational costs of running many SVM model with high dimension vectors slightly limited the training on the range of hyperparameters. Implementation limitations were also a factor: kernel design in Python still has room for improvement, and kernel closure properties have to be computed manually. As of today (2022), there are no available frameworks for Deep Kernel Learning, and combinations/compositions of functions also have to be calculated manually.

In Transfer Learning, instead of using a single learning rate for top layer training an another one for training updating all weights, differential learning rates could be considered. It considers assigning lower learning rates to convolutional blocks closer to the input images, as these tend to hold more

fundamental and generalizable latent concepts. As mentioned earlier, the method could significantly benefit from trying more top-layer architectures and/or different convolutional bases.

An important next step in understanding how the model classifies the images is to look into interpretability of Convolutional Neural Networks. This fell out of the scope of the project but could deliver valuable insight into the discussion of what characterizes a particular style.

References

[1] A. Pladevall, *This is Catalonia : guide to the architectural heritage*. Generalitat de Catalunya, Departament de Cultura, 1993.

[2] G. de Catalunya. Departament de Cultura i Mitjans de Comunicació, "Patrimoni.gencat."

[3] O. Chapelle, P. Haffner, and V. Vapnik, "Support vector machines for histogram-based image classification," *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 10, pp. 1055–64, 09 1999.

[4] G. Camps-Valls, L. Gomez-Chova, J. Munoz-Mari, J. Vila-Frances, and J. Calpe-Maravilla, "Composite kernels for hyperspectral image classification," *IEEE Geoscience and Remote Sensing Letters*, vol. 3, no. 1, pp. 93–97, 2006.

[5] F. Sultana, A. Sufian, and P. Dutta, "Advancements in image classification using convolutional neural network," *CoRR*, vol. abs/1905.03288, 2019.

[6] B. Xia, X. Li, H. Shi, S. Chen, and J. Chen, "Style classification and prediction of residential buildings based on machine learning," *Journal of Asian Architecture and Building Engineering*, vol. 19, no. 6, pp. 714–730, 2020.

[7] Y. Yoshimura, B. Cai, Z. Wang, and C. Ratti, *Deep Learning Architect: Classification for Architectural Design Through the Eye of Artificial Intelligence*, pp. 249–265. Cham: Springer International Publishing, 2019.

[8] S. S. Du, Y. Wang, X. Zhai, S. Balakrishnan, R. R. Salakhutdinov, and A. Singh, "How many samples are needed to estimate a convolutional neural network?," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[10] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 2016.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[12] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," 2016.

[13] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2019.

[14] G. LoweDavid, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, 2004.

[15] E. Karami, S. Prasad, and M. Shehata, "Image matching using sift, surf, brief and orb: Performance comparison for distorted images," 2017.

[16] I. Otero, *Anatomy of the SIFT method*. PhD thesis, 09 2015.

[17] J. Wilson and M. Arif, "Scene recognition by combining local and global image descriptors," 2017.

[18] V. Chauhan, K. Dahiya, and A. Sharma, "Problem formulations and solvers in linear svm: a review," *Artificial Intelligence Review*, vol. 52, pp. 803–855, 08 2019.

[19] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[21] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *Journal of Machine Learning Research*, vol. 9, 2008.

[22] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[23] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The Annals of Statistics*, vol. 36, no. 3, pp. 1171 – 1220, 2008.

[24] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[25] M. Gönen and E. Alpaydin, "Multiple kernel learning algorithms," *Journal of Machine Learning Research*, vol. 12, no. 64, pp. 2211–2268, 2011.

[26] M. Arbel, J. Mairal, and J.-P. Vert, "Machine learning with kernel methods. course slides,"

[27] I. Lauriola and F. Aiolli, "Mklpy: a python-based framework for multiple kernel learning," *arXiv preprint arXiv:2007.09982*, 2020.

[28] F. Aiolli and M. Donini, "Easymkl: a scalable multiple kernel learning algorithm," *Neurocomputing*, vol. 169, pp. 215–224, 2015. Learning for Visual Semantic Understanding in Big Data ESANN 2014 Industrial Data Processing and Analysis.

[29] E. V. Strobl and S. Visweswaran, "Deep multiple kernel learning," in *2013 12th International Conference on Machine Learning and Applications*, IEEE, dec 2013.

[30] J. Zhuang, I. W. Tsang, and S. C. Hoi, "Two-layer multiple kernel learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (G. Gordon, D. Dunson, and M. Dudík, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 909–917, PMLR, 11–13 Apr 2011.

[31] B. Bohn, M. Griebel, and C. Rieger, "A representer theorem for deep kernel learning," *CoRR*, vol. abs/1709.10441, 2017.

[32] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," 2017.

[33] Keras, "Transfer learning and fine-tuning nbsp;: nbsp; tensorflow core."