Artwork recognition (based on the iMet Collection 2019 competition)

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

Abstract

In this work we explore some common tasks of modern Computer Vision. We apply classic machine learning algorithms with bag-of-words descriptors and state-of-the-art deep learning architectures to a multi-class classification problem of objects in a Museum (the Met). Since the dataset is highly unbalanced dataset, we explore techniques such as data augmentation. The architectures we use are then adapted to use in a multi-label classification problem. Finally we explore the usage of CAMs (class activation mapping) to tackle a painting object detection task.

1 Introduction

In this work we created a system for automatic classification of artwork from images, all of them containing artefacts which are part of the collection of The Metropolitan Museum of Art in New York. The dataset used in the project was acquired from the iMet Collection 2019 competition. Each image in the complete dataset has at least one attribute label from a label set of 1103 attributes. The work consists of three tasks each one addressing one of the main problems in computer vision, namely multi-class classification, multi-label classification and object detection, which will be described in detail.

2 Task 1: Multi-class classification

In machine learning, the task of assigning a single label to an input is called classification. The dataset we used is a subset of the original dataset from the Metropolitan museum. It contains only 50 possible classes and is highly unbalanced, as can be seen in figure 1, which is a factor that needed to be addressed in the implementation. We used two different methods to achieve multi-class classification, one being a more classical approach that relies on creating a visual vocabulary for feature representation, and the other being a deep learning method, specifically convolutional neural networks.

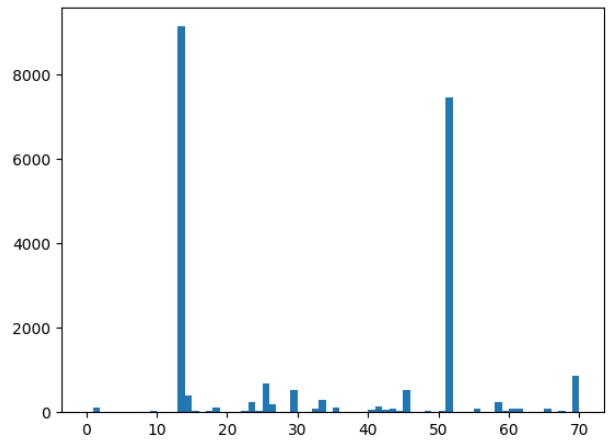

Figure 1: Class distribution for the reduced dataset. Classes 13 and 51 account for more than two thirds of the dataset.

2.1 Dictionary-based representation: BOW + SVM

2.1.1 Bag-of-words

Influenced by text analysis, the bag-of-words model is based on a dictionary of visual words, each representing an image feature. Keypoints are extracted from training images and then grouped into a number of clusters. By treating each cluster as a visual word, we obtain a visual word vocabulary describing all kinds of local image patterns. An image can then be represented as a feature vector containing the count of each visual word in the image.

Vocabulary size plays a key part in the success of the bag-of-words model. A small vocabulary groups dissimilar keypoints into the same cluster, resulting in visual words that are not representative of all present patches. Although a large vocabulary tends to improve matching accuracy, it being too large can lead to overfitting in the classification step. Keeping this in mind, we created vocabularies of three different sizes, namely 500, 1000 and 2500, in order to empirically determine the most fit one for this specific task.

Model	Accuracy	Precision	Recall
V_{size}:500,C:10,gamma:scale	0.578	0.549	0.578
V_{size}:1000,C:10,gamma:scale	0.581	0.544	0.581
V_{size}:2500,C:1,gamma:scale	0.563	0.482	0.563

Model	AUC_{ROC}	Top 5 Acc.
V_{size}:500,C:10,gamma:scale	0.878	0.922
V_{size}:1000,C:10,gamma:scale	0.863	0.924
V_{size}:2500,C:1,gamma:scale	0.842	0.910

Table 1: BoW + SVM results

2.1.2 SVM

Support vector machines, SVM's for short, are classification models based on finding the maximum margin between two classes. The margin can either be hard or soft, the latter allowing examples to pass the margin border and being penalized. This addition allows for accurately classifying inputs which are not linearly separable, as well as responding better to noise. Two hyperparameters regulating the strictness of the border and thereby how loose the classification is are C and gamma. We optimized these by using grid search, choosing 0.1, 1 and 10 as the possible values for C and 'auto' and 'scale' values for gamma, meaning either the reciprocal value of the number of features or of the product of the number of features and the variance of the input. We selected RBF as the fixed kernel type, the reason being it's one of the most successful for usage in SVM. Since SVM is inherently a binary classifier, a method for multi-class classification needs to be adopted. We opted for the *one-vs-rest* approach because of a big number of classes.

2.1.3 Base results

The comparison of vocabulary sizes and SVM parameters is pictured in Table 1. No big differences in performance between the vocabulary sizes of 500 and 1000 are visible, however, the vocabulary of 2500 words seems to be achieving slightly worse results. Considering the classes were highly unbalanced, the results are according to our expectations. We chose to address the problem of unbalanced classes in the deep learning approach since we had more hope of achieving greater performance using that method. Possible improvements of this model would include a broader search for vocabulary sizes and grid search parameters, which is computationally expensive and was therefore hard to execute.

2.2 Deep learning methods

This section presents our approach at applying deep learning methods to solve the proposed classification task. One of our concerns when choosing which network architectures to use was the number of trainable parameters, since this has great influence in the performance of the network.

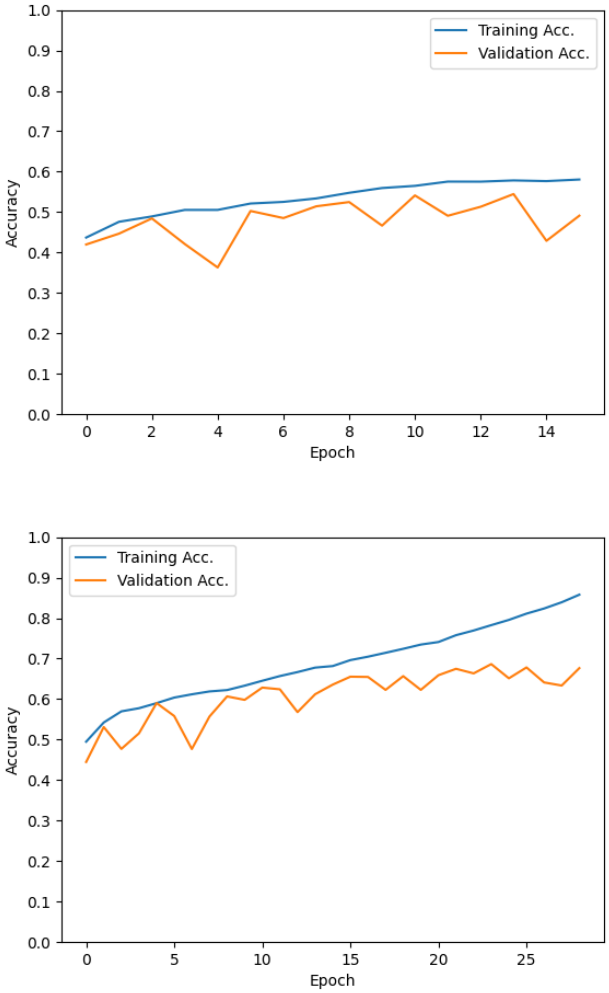

Figure 2: Test (blue) and validation (orange) accuracy for the AlexNet (top) and InceptionV3 (bottom) networks. The number of epochs is different due to the early stopping mechanism.

As such, and taking into consideration the size of our dataset, we decided to use AlexNet [3] and InceptionV3 [5]. While the first has roughly 60 million parameters, the latter only has around a third of that. Consequently, during our experiments we found that training AlexNet was much faster than training the InceptionV3 net. Additionally, we implemented the AlexNet as per the paper's specification using the Keras API whereas we uses the InceptionV3 model available in Keras.

2.2.1 Base results

We trained both networks for a maximum of 100 epochs, with an early stopping mechanism once the validation loss didn't improve for 5 consecutive epochs. We used a batch size of 32, a categorical cross entropy loss (which is suitable for multi-class classification) and the Adam optimizer [2] using the default parameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-7}$ and $learning\_rate = 0.001$). We recorded the Top1 accuracy, Top5 Accuracy, Precision, Recall and Area under curve for each epoch. In order to properly train the networks we split the dataset in two, a training set and a test set. Furthermore, when training, 20% of the set is used for validation and the rest is given as input to the network. It is also important to mention that all of the images in the validation and test sets are original. Figure 2 presents the evolution of the test and validation accuracy through the epochs for the AlexNet and InceptionV3 networks.

As we can see, the Inception network was able to obtain an higher validation accuracy of 0.68 compared to AlexNet's 0.54 which is to be expected, since the Inception network is both more complex and recent than the AlexNet, however training the Inception network took almost double the amount of epochs. As we can see in table 2 (rows *incep_4* and *anet_4*), the test accuracy for these models is 0.689 and 0.546 respectively. The other metrics were higher using inception, with the starkest

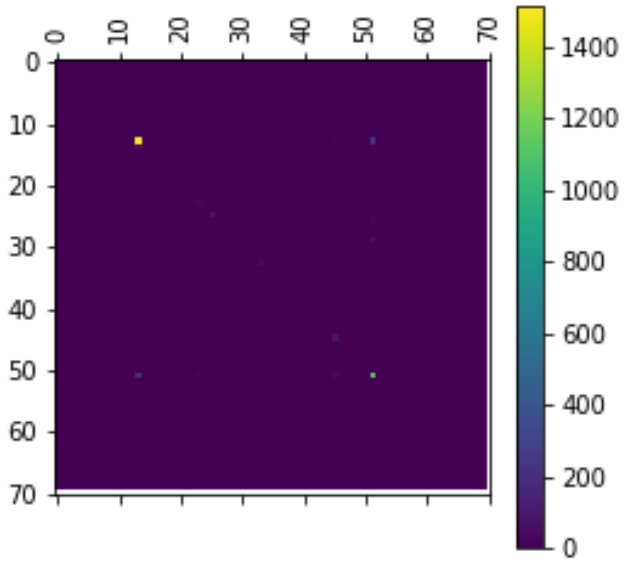

Figure 3: Confusion matrix for the InceptionV3 (incep_4) network using the test set.

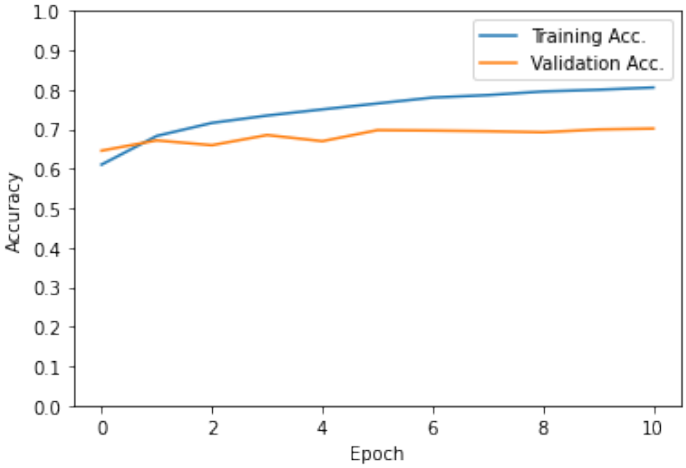

Figure 4: Test (blue) and validation (orange) accuracy for the InceptionV3 network.

difference being the Recall, this means that this network is able to return more relevant results.

We must take these results with a grain of salt, since the dataset is so unbalanced. As we can see in the confusion matrix for InceptionV3 in figure 3, most of the true positives happen for guessing images with labels 13 and 51, which account for two thirds of the dataset (which is similar to the accuracy).

2.2.2 Transfer learning

In order to explore the technique of transfer learning, we took advantage of the fact that we were using the Inception model from the *Keras* library, which has the option of instantiating the network with weights calculated by training the network with the *Imagenet* dataset. By freezing the convolutions layers' weights, we were essentially using a pre-trained feature extractor (the convolutional layers of the InceptionV3 network) and training the classifier on the iMet dataset.

We used the same training setup described in the previous section. Figure 4 presents the evolution of the test and validation accuracy through the epochs for InceptionV3 networks with transfer learning.

As we can see, this approach presents slightly better results than the previous one, and is able to achieve these results very early, and doesn't present much evolution through the epochs. The testing accuracy of this approach was slightly higher than the previous one at 0.696. In fact, as

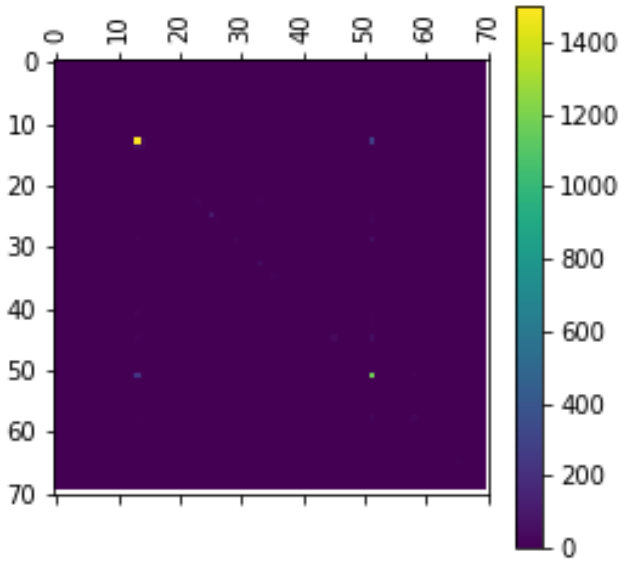

Figure 5: Confusion matrix for the InceptionV3 (*incep_3*), with transfer learning, network using the test set.

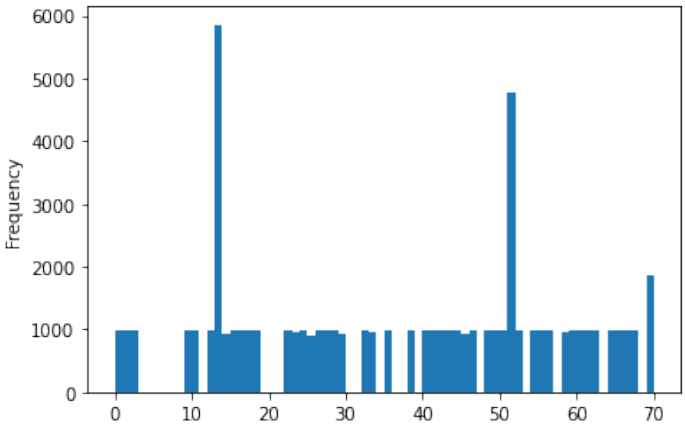

Figure 6: Class distribution for the augmented dataset.

can be seen in table 2, every metric was better (slightly) in the transfer learning (incep_3).

Despite having slightly better results, we can still attribute most of the performance to the unbalanced nature of the dataset. This is once again evidenced by the confusion matrix in figure 5, which shows that the main source of correct predictions comes from the most present classes (13 and 51).

2.2.3 Data augmentation

As shown in the previous sections, most of the networks performance can be explained by the unbalanced nature of the dataset. We tried to tackle this by performing data augmentation on the dataset.

In order to augment the dataset we took advantage of the *ImageDataGenerator* object provided by Keras, which allowed us to create images based on images of the dataset, where a set of transformations were applied. With this we implemented the augment_dataset function, which stores the generated images and returns the image to label mapping. The image generation is achieved by applying modifications to the original images. We tried to choose transformations that wouldn't be large deviations from the real data, for example, it wouldn't make sense to two vertical flips to the images. As such the chosen transformations are: flipping the image horizontally, making it brighter of darker, zooming in or out, translating and rotating by small amounts. The values for all these modifications are randomly generated within their own ranges.

We tested the augmentation on the transfer learning InceptionV3 network, since its the best one we had. We chose to augment each class to have at least 1000 images. In the end, the class distribution was the one presented in figure 6. The experimental setup was the same explained in the previous section. Figure 7 presents the evolution of the test and validation accuracy through the epochs for the network.

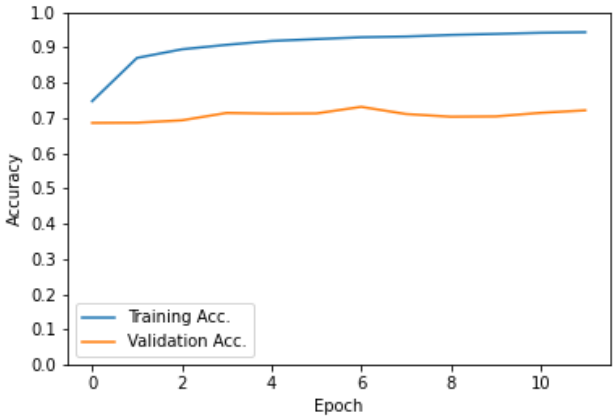

Figure 7: Test (blue) and validation (orange) accuracy for the InceptionV3 network using transfer learning and data augmentation.

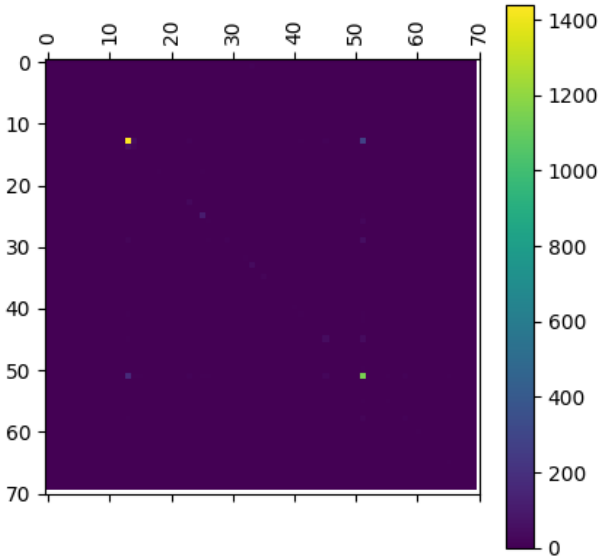

Figure 8: Confusion matrix for the InceptionV3 (*incep_1*), with transfer learning and data augmentation, network using the test set.

Despite achieving a higher test accuracy, compared to the approach presented in the previous section, using data augmentation achieved about the same validation accuracy. In addition, looking at the testing results on table 2, we can see that every metric was worse than the non-augmented counter-part.

In fact, by inspecting, once again, the confusion matrix (figure 8), we are able to see that the main source of accuracy for this network is still due to the two most prevalent classes (13 and 51). We attribute the failure of the augmentation due to the fact that most of the classes had very few examples in the original dataset, so the augmentation to 1000 didn't produce enough variety to allow the network to really learn to discriminate between those classes.

3 Task 2.1: Multi-label classification

The task described in this section differs from the aforementioned because in a multi-label setting each example can belong to multiple classes while in multi-class each example belongs to exactly one class.

Model	Accuracy	Precision	Recall	AUC_{ROC}	Top 5 Acc.
anet_4	0.546	0.662	0.391	0.960	0.897
incep_1	0.688	0.729	0.658	0.972	0.962
incep_3	0.696	0.754	0.638	0.980	0.960
incep_4	0.689	0.740	0.639	0.968	0.938

Table 2: Test results for multiclass classification. The row labelled as *anet* is for the AlexNet whereas the *incep* labels refer to the InceptionV3 network. *incep_4* and *anet_4* represent the base implementations. *incep_3* and *incep_1* use of transfer learning. *incept_1* uses data augmentation.

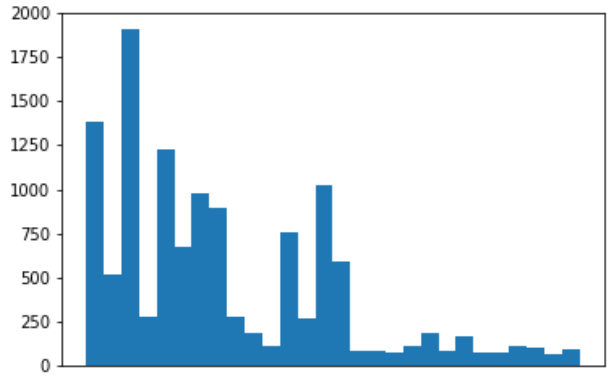

Figure 9: Number of examples for each label in the smaller dataset.

A possible solution to address this is to adapt the CNN networks used in the previous task. By using softmax activation in the last layer, the networks used in the previous task output a probability distribution with the probability of the image belonging to each class, and the image belong to the class with the highest probability. Since in this task an image can belong to multiple classes, we replace the softmax activation with a sigmoid activation, so that each label is assigned with an independent probability. Then, if the output for a given label is higher than a specified threshold, then that label is true for that image. By using sigmoid, an image can belong to any combination of labels. The used loss function is Binary Cross Entropy, instead of Categorical Cross Entropy.

Similarly to the previous task, the dataset used is highly unbalanced, as can be seen in Figure 9. Because of hardware limitations, we used the smaller subset provided by the professors. The training setup is similar to the one in the previous task, with the same optimizer. Again, we extracted 20% of the set for testing. Of the training set, we used 20% for validation.

Since the Inception architecture achieved the best results for the previous task, we used the it for Task 2. We trained an Inception network from scratch (referred to as *incep_4*). We also used a transfer learning approach with a network pre-trained on ImageNet (referred to as *incep_3*). Figure 10 and Figure 11 show the evolution of the F2-score, considering a threshold of 0.5 for decision, on the train and validation sets.

We evaluate the model using the F2 measure, as in the original competition. This measure is the general F score (Equation 3 with β equal to 2, so that recall is twice as important as precision. The presented F2 score is obtained by averaging the F2 score for each example. We present the F2 score using different decision thresholds. We also measure the Area Under Curve (AUC) for the ROC and PR curves.

$$F_\beta = (1+\beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \quad (1)$$

Model	AUC_{ROC}	AUC_{PR}	$F2_{0.2}$	$F2_{0.3}$	$F2_{0.4}$	$F2_{0.5}$
incep_3	0.847	0.340	0.448	0.363	0.297	0.237
incep_4	0.838	0.342	0.433	0.354	0.277	0.219

Table 3: Performance metrics obtained on the test split of the dataset using the Inception architecture without transfer learning (*incep_4*) and the Inception architecture with transfer learning (*incep_3*)

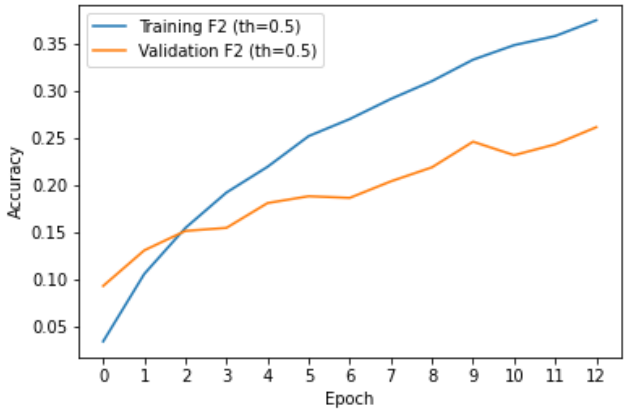

Figure 10: F2 score (with a threshold of 0.5) for the validation and train sets, using an Inception V3 net.

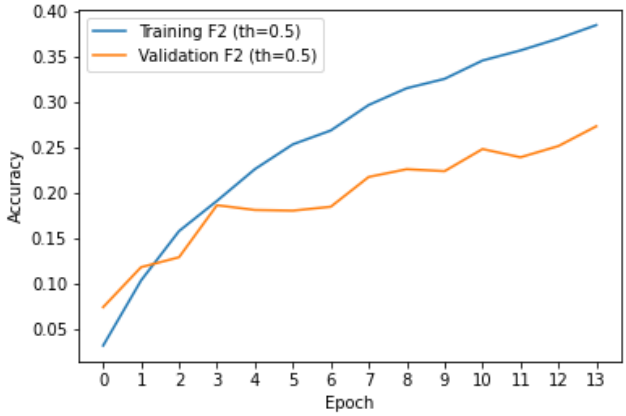

Figure 11: F2 score (with a threshold of 0.5) for the validation and train sets, using an Inception V3 net pre-trained on ImageNet.

In Table 3 we present the metrics obtained on the test set. We see that the model with transfer learning performed slightly better than the one without it, except for the area under the precision and recall curve metric. We can also see that using a lower threshold for considering a label as positive/negative also resulted in a better F2 score.

4 Task 2.2: Artwork detection

The goal of this task is to detect paintings in a picture. In order to regress bounding boxes on the positions of paintings in an image, we would need a dataset of images containing labeled bounding boxes surrounding paintings to train any any object detection algorithm e.g. YOLO. Since we don't have such a dataset, we could create our own dataset, by artificially creating images of rooms containing paintings on the wall, and labelling them with the correct bounding boxes.

Another option was to investigate the possibility of using CAM (Class-activation map) methods to make the object detection of the paintings based on a classifier. Due to the short time we had for this project, and group's general curiosity towards this technique, we opted on this approach. On account of the change in the specification, we focused our efforts on Task 2.1, however, we'll still present our rationale and preliminary results in this section.

The goal of class activation maps, as described in [6], is to use the convolutional layers of a CNN in order to detect objects by displaying the regions of the image that maximally activate them, and affect most the network's classification – a technique of **weakly-supervised object localization**. Then by calculating a box through the use of a threshold on the heat map the technique is able to localize the objects without having to train a network specifically for that purpose. What follows is the steps we took to implement the technique.

4.1 Classifier training

In order to implement the CAM algorithm, we needed a classifier that would be able to classify paintings. We settled on training a classifier to distinguish between images of paintings and sculptures, using for the effect part of the *Art Images* dataset [1]. Later, we'll discuss why this may not have been the best approach.

We used a *VGG-19* [4] network, that was modified to remove the classifier (3 fully connected layers at the top of the network), and the last convolutional/pooling layers (until layer *block5_conv3*), which we replaced by 1024 filters (2D convolutions) with a 3x3 kernel followed by a global average pooling layer connected to a 2-unit fully connected layer with a soft max activation function (one unit for each class). The removal of the last few layers of the feature-extracting section of the network is done in order to increase the spatial resolution of the layer that "fed" the GAP layer, which ended up being 14x14 for the *VGG-19 network*. In summary, the VGG-19 "sub-net" outputs a 14x14x1024 volume which is then condensed into a 1x1x2014 volume by the GAP layer, which in turns serves as an input to a fully connected layer outputting two units activated by a softmax function.

We trained the network on a dataset containing 3780 images of paintings and sculptures (2042 paintings and 1738 sculptures) and validated the model on a set of 419 images (228 paintings and 192 sculptures). We ran the model for 10 epochs with an early stopping oriented by the validation loss (with patience equal to 2). Figure 12 shows the results of the training

The classifier achieved excellent results in this task peeking at about 94.9% for the validation accuracy. This is to be expected since we have a powerful and large network, coupled with a relatively small dataset, composed of two very distinct classes.

4.2 CAM

The class activation maps were implemented as described in the paper. Each channel of the 14x14x1024 volume resultant of the application of the aforementioned 1024 filters were summed, each weighed by the weight of the connection between the corresponding unit of the GAP output and the network output layer unit of the selected class. This is better explained by Figure 13, which we took from the original paper. Furthermore, the code segment that calculates the CAM for the selected class can be seen in source code 1. The resultant matrix was then normalized, scaled and

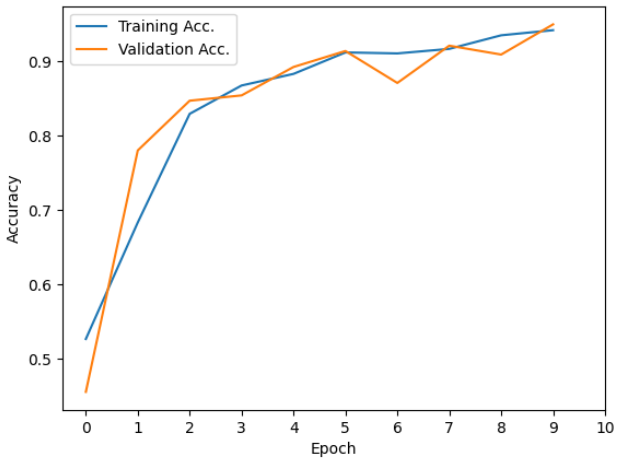

Figure 12: Training results for the modified VGG-19 network classifying images between painting and sculpture. The blue and orange lines represent the training and validation accuracy per epoch, respectively.

added to the original image in order to evaluate which regions were most responsible for the activation.

Source code 1: CAM implementation

```
def returnCAM(feature_conv, weights):
    h, w, nc = feature_conv.shape

    # Multiply each feature map with it's
    # importance for the classification
    reshaped_feature_conf = feature_conv.numpy()
        .reshape((h*w, nc)
    cam = np.matmul(reshaped_feature_conf),
        weights)
    cam = cam.reshape(h, w)

    # Normalize the heatmap
    cam = cam - np.min(cam)
    cam_img = cam / np.max(cam)
    cam_img = np.uint8(255 * cam_img)
    return cam_img
```

Since are only looking for a qualitative evaluation of the method, we decided to include a wide variety of test scenarios.

First, as a control, we tested scenarios with images from the dataset (figure 14), where can see that, the labels were correctly attributed, with an higher confidence for the sculpture. In fact, the network seemed to have a higher confidence when faced with images from sculptures, this may be because sculptures are a less diverse domain than paintings (at least in the used dataset).

Using images from outside (figure 15) the domain we can see that the network was not able to identify the painting correctly, which we attribute to the fact that the images of paintings that the network trained on were paintings that covered the full image, instead of being paintings in a different context, e.g. a framed painting in a room.

Figure 16 contains the type of images that are the subject of this task, and we can see that not only the network was unable to correctly identify some of the inputs, even when it succeeded, the classification wasn't made due to the actual paintings. These activation maps can not be used for object detection.

We decided to also test images of rooms containing sculptures in order to access if the bad results were exclusive to the paintings class (which is relevant since the domains are very different). This time, the network was able to identify the images correctly with an high confidence. Despite this, we were still unable to achieve good results in detecting the several sculptures present in the room. However, in this case the attention of the network seemed to be able to find part of the sculptures, particularly the head of the central sculpture in the second image, and the top right sculpture in the last image.

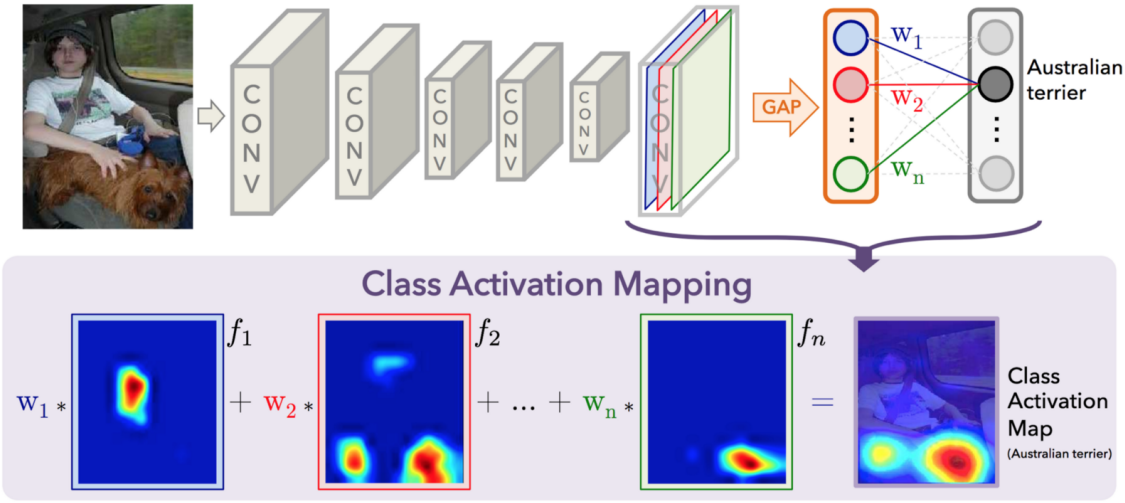

Figure 13: Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions. As taken from [6].

Figure 14: CAM results on images from the dataset

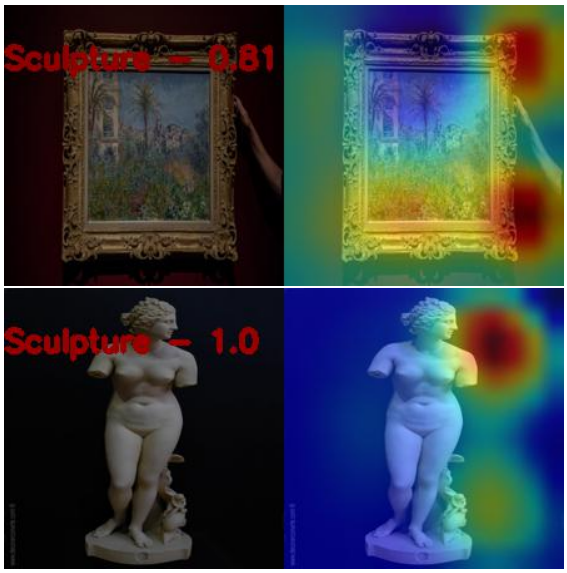

Figure 15: CAM results on images of the domain that are not in the dataset

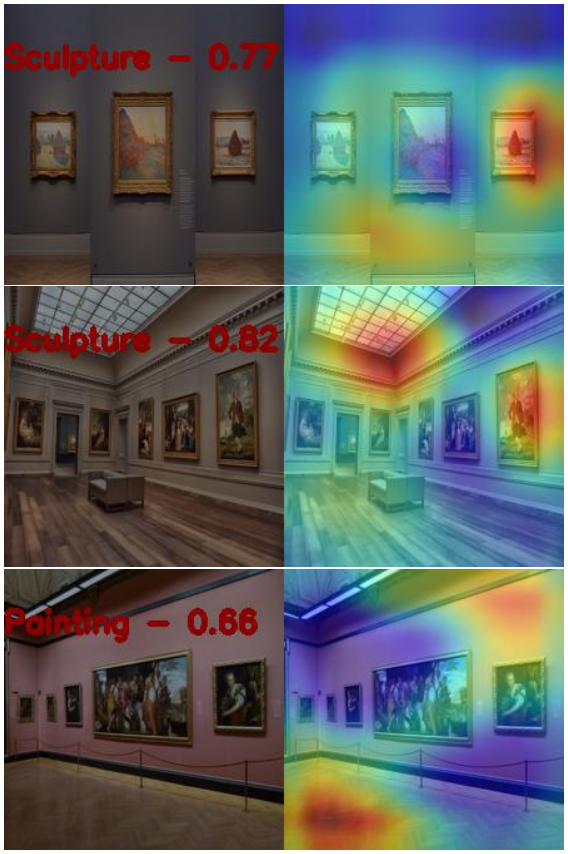

Figure 16: CAM results on images from paintings in museums (task goal)

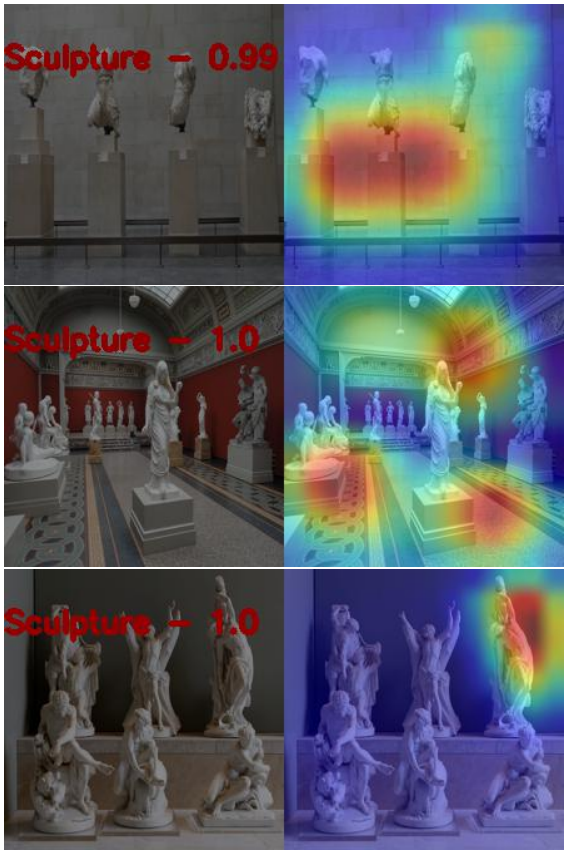

Figure 17: CAM results on images from sculptures in museums

4.3 Discussion

This section will discuss issues with our approach and implementation, and what we would've made differently if we had more time. Firstly, we intended to test our implementation of CAM in the same dataset used in [6], or in another dataset where CAM as been successfully used, in order to prove the correctness of the implementation. Secondly, we conclude that our choice of the dataset wasn't the best, because the classifier is trying to discriminate between two distinct domains, whereas if we used a dataset where the network at to classify the image between different painting types, authors, styles, etc., the network would be more specialized in finding paintings. Furthermore, we'd advocate that the method would have more success if the images were images of paintings in context i.e. the framed painting in a wall, instead of being images of just the painting itself. By training the network to classify paintings in a context, we speculate that it wouldn't be such a stark difference between those images and the ones of paintings in a museum room.

5 Conclusion

This work contains a good display of the acquired knowledge during the 2^{nd} part of the VCOM curricular unit. It contains our rationale and attempts at problem solving in order to improve the a good performance in the proposed tasks. Although the results were less then optimal, we still argue that we were able to identify problems and propose solutions to tackle them.

To this extend, if we had the opportunity to continue the work, we would like to test the techniques using the full datasets, as to get a better estimate of the performance of the networks without being hindered by class unbalancing. Especially in Task 2.1, due to constraints in time we were not able to properly address the class unbalance issue. We would like to explore if using with different thresholds for each label (instead of using the same threshold for all labels, as described in this report) could be fruitful to help with the unbalance issue.

In addition, we would change our approach in task 2.2, and try to find a more suitable dataset of paintings for training the classifier.

References

[1] Danil. Art images: Drawing/painting/sculptures/engravings, May 2018. URL https://www.kaggle.com/thedownhill/art-images-drawings-painting-sculpture-engraving.

[2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[4] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[5] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016. doi: 10.1109/CVPR.2016.308.

[6] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2921–2929, 2016. doi: 10.1109/CVPR.2016.319.