Artwork Recognition

I. Introduction

This article has the goal to explore different methods and architectures related to image classification and labeling. Making use of various state of the art techniques for classification of different classes and labels for images and handling highly imbalanced data, poor class representation and noisy data. Convolutional Neural Networks are widely used in this context but there are alternatives and a vast amount of techniques domain-dependent that can enhance our results. Here we will explore some and retain conclusions from our experiments.

II. State of the art

A. Feature extraction

Some years ago, the spatial pyramid representation with a feature extractor, such as SIFT, was commonly used to extract visual features. However, the performance was far from excellent. Later, the convolution neural networks were introduced and they are still used until this day and seen as the main way to extract features from an image and hence the state-of-the-art classifiers also used them in an initial phase. These networks learn the filters that should ”scan” the input to create feature maps and hence summarize the image by finding its main features. Throughout the years, inside this field, there was the elaboration and construction of new architectures that tried to mitigate the flaws of the older ones, bringing to the table new techniques to increase efficiency and efficacy. For instance, with VGG there was introduced the dilemma of ”deep vs wide”, as a way to diminish the number of trainable parameters without hurting the performance. ResNet50 architecture introduced skip-connections that lead to great improvements and allowed the architecture to be very deep. Another example would be the inception architecture that tries to perform feature extraction in different scales and joins everything together to get better values at each building block and, in the end, uses global pooling as a way to reduce the number of parameters that the FC needs to contain. Nowadays there are even architectures that join these two into one -Inception-ResNetV2. Also, newer versions of the older architectures are emerging as an improved version of them as we can easily see in their name at the end. These high-performance architectures are quite well-known and there are versions of them available to the public that were previously trained on huge datasets allowing programmers to perform a very much used technique nowadays called transfer learning.

B. Transfer learning

Transfer learning is a machine learning technique frequently used to leverage knowledge gained about feature representation while solving another problem and applying it to a different but related problem. Moreover, training deep networks on small datasets (one that is smaller than the number of parameters) greatly affects the network’s ability to generalize, often resulting in overfitting. So, this works when pre-trained models are trained on massive datasets such as ImageNet. This way, the pre-trained model already learned to capture universal features, such as curves, color gradients, and edges in its early layers, which can be helpful to a great number of computer vision classification problems. Therefore, this technique is widely used nowadays, especially when people do not have enough data and resources. Additionally, there are two options when doing transfer learning. The parameters of the last layers are the only ones that are updated or all model’s layers can be updated. The first option is often referred to as feature extraction, while the second is referred to as fine-tuning.

C. Classification

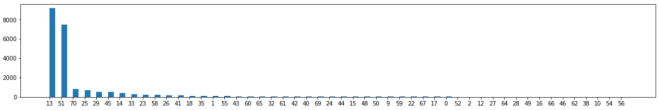
A classifier is any algorithm that sorts data into labeled classes or categories. Since nowadays there is a lot of data available in many fields, deep learning models are often used for classification. These models are bound to work and generalize well and hence are the state-of-the-art. Nevertheless, when less data is available, some classical classification models, such as Support Vector Machines can be used.

III. Datasets

A. Multiclass dataset

The dataset used in this report contains a large number of artwork images of art pieces from The Metropolian Museum of Art of New York for the iMet-2019 competition. However not the entire dataset was used, two files containing a subset of images and the associated classes/labels was supplied. The objects in the image tend to be centered in the middle and there are no assure regarding the correctness of the classes and labels provided, so the data must be treated as noisy. Images come in different sizes so a resize must be done in order to standardize data.

Regarding the multi-classification file, there are a total of 22087 entries and 50 classes, from which about 75,2% of the entries correspond to either class 13 or 51, 9151 and 7463 entries respectively, and 40 classes with a representation less or equal to 1% of the total dataset, some even with less than 0.1%. As one can see the data is highly imbalanced, with two classes dominating the dataset and considerable part with a poor representation.



{13: 9151, 51: 7463, 70: 831, 25: 676, 29: 530, 45: 514, 14: 408, 33: 292, 23: 244, 58: 240, 26: 177, 41: 141, 18: 121, 35: 119, 1: 100, 55: 91, 43: 74, 60: 74, 65: 72, 32: 72, 61: 71, 42: 50, 40: 48, 69: 43, 24: 41, 44: 35, 15: 34, 48: 34, 50: 29, 9: 28, 59: 22, 22: 22, 67: 22, 17: 20, 0: 18, 52: 17, 2: 14, 12: 14, 27: 13, 64: 13, 28: 12, 49: 12, 16: 12, 66: 11, 46: 11, 62: 11, 38: 10, 10: 10, 54: 10, 56: 10}

Fig. 1: Multiclass data frequency

B. MultiLabel dataset

The dataset used for the multilabel classification is similar to the one used for the multiclass classification, being composed from the same artwork images from the Metropolitan Museum of Art of New York. Two files detailing relations between images and their respective sets of labels, one containing information pertaining to the whole dataset, and another containing only a subset of the original. In the larger dataset, the training set was composed of 74885 images, and the validation set was composed of 18721 images, both of which identified 72 labels. The smaller set's training set was composed of 9546 images, and the validation set was composed of 2386 images, having identified 28 labels.

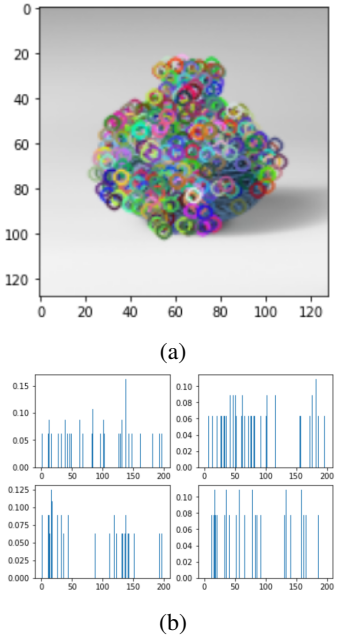
IV. Multiclass Classification

A. Dictionary-based representation & Classifier

The first approach for solving this problem was to use a Dictionary-based representation, more specifically Bag of Visual Words in conjunction with a Classifier. The first step in this strategy was to split the dataset into training and testing. Given that this dataset is heavily unbalanced it was decided to modify the training dataset. After analyzing several options to solve this type of problem it was decided to use Over Sampling in a slightly different way than usual. The strategy used consists consists of using Data Augmentation to create new images similar to images of an infrequently used class. These new images were added to the training dataset with the same class as the image from which they were generated, thus increasing the frequency of the respective class.

Fig. 2: Data Augmentation for image with infrequently used class

Having applied the strategy to solve the mentioned problem, the next step is implementing the Bag of Visual Words (BoVW) and the classifier. First of all, the features and their respective descriptors are extracted from each image in the dataset. For this, we can use a feature extractor algorithm, such as BRISK, the feature extractors we used in our solution, which is a good alternative to the SIFT feature extractor, with slightly faster extraction periods. After obtaining the descriptors we used a Pipeline together with GridSearchCV that allowed us to get the best hyperparameters for the different steps in the pipeline. The first step of the pipeline is responsible for the BoVW, by starting with the clustering of the training image descriptors to obtain the visual vocabulary and afterwards calculating the histogram for the images. The clustering was implemented using MiniBatchKMeans algorithm which corresponds to a faster version of the KMeans algorithm. To perform clustering the number of centroids must be chosen cautiously in order not to give overfit or leave features out, this choice was easier since using GridSearchCV allowed us to determine which value gives us the best results. After performing clustering, we have the visual vocabulary and therefore we can calculate the different bag of words in which each image will be represented by a histogram. BoVW is simple and efficient, but it discards information about the spatial structure. To preserve the spatial information we used spatial pyramid matching, where it is calculated the histogram values for several levels, more specifically two levels. For each level we chop the image into $2^l * 2^l$ cells where l is the level number. We treat each cell as a small image and count how often each visual word appears, thus obtaining the histogram. After this, the histograms of each level are normalized since words can have more importance than others, and at the end the different histograms are merged together. This merging is done by weighting more histograms obtained at finer resolutions, i.e. giving more weight to the histograms obtained at the levels with more image divisions, thus giving more importance to the spatial information.



(a)

(b)

Fig. 3: (a) Keypoints obtained with BRISK (b) Spatial Pyramid histograms for level 2

The next step of the pipeline is an intermediate step where StandardScaler was used to normalize the input for the classifiers. This means scaling the input so that their values fall more or less in the same range. The last step of the pipeline consists of classifying the images, where initially the classifier is trained on the training dataset and then the classes from the test dataset are predicted. In this step several classifiers were used in order to understand the different behaviors. The classifiers used were a SVC (Support Vector Machine), a RandomForestClassifier and a LogisticRegression and were used together with the GridSearchCV in order to try to obtain the best hyperparameters. A very important aspect to note regarding the training of these classifiers was the use of cross validation, more precisely of StratifiedKFold, where the folds are made by preserving the percentage of samples for each class. This is important since the dataset is unbalanced and this could lead to using training folds with missing classes, which would lead to bad results. Another important

aspect to highlight regarding the classifiers was the use of the *class_weight* parameter. This is again important because the dataset is unbalanced, since this parameter will affect the loss calculation to penalize differently a false classification of the minority and majority class. In this case the *balanced* value was used so that the weight applied is inversely proportional to the class frequency. A final point to note was the use of OneVsRestClassifier encapsulating the classifiers used. This method is widely used for multiclass classifications, since it splits this classification into several binary classifications, thus allowing the classifier to make better decisions. Furthermore, the OnevsOneClassifier was also used for the SVM classifier, this method divides the different classes into pairs and applies a binary classification. This method was used for this classifier so that it could better understand the differences between the different classes, since this classifier predicted in most test images the classes with higher frequency. Regarding the results, the following tables show the results obtained using the different classifiers.

Model	Accuracy	F1 micro	F1 weighted
RandomForest	20%	19.6%	23.3%
SVM OneVsRest	48%	49.3%	43.1%
SVM OneVsOne	48.7%	49.9%	43.5%
Logistic Regression	23.9%	23.1%	28.4%

TABLE I: Metrics using BRISK feature detector

As shown above, the classifier that obtained the best results was the SVM. We can state that both the accuracy and the f1 score have positive values that could lead us to use this solution to solve this problem, but analyzing in a more appropriate way, more precisely evaluating the confusion matrix we can realize that the classifier did not perform well since it predicted most of the time the classes 13 and 51 that correspond to the classes with higher frequency, not being able to really distinguish the different classes. The other two classifiers however did not show the same behavior, since the confusion matrices show that there is a wider distribution in the class choices. Despite this, they did not show very positive metric values, which may reveal that the features extracted for these images and the respective construction of the visual vocabulary were not very effective.

B. Convolutional Neural Network

In a second approach we opted for a Convolutional Neural Network (CNN).

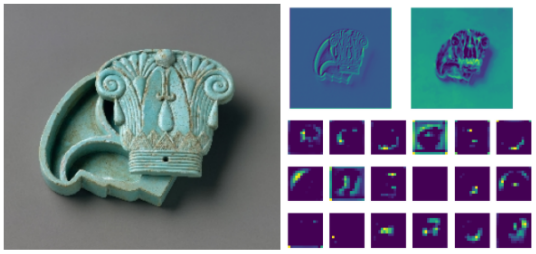


Fig. 4: Image 10943defdd5d5e89 and some of its activation maps in the second and last convolutional layers

The CNN tends to produce great results with image applications, specially where feature extraction and classification are the targets. It turns out this is one of those problems. One issue with our data is that it is highly imbalanced with some classes having less than 0.1% of representation as described in a previous section however CNN needs a lot of data in order to accurately be able to extract relevant features, so work was developed in that direction. Also, with the help of Keras *ImageDataGenerator*, during the training and validation phase, the training data passed to the model was augmented in order to make a more robust and general model and also improve representation of low frequency classes. The augmentation was made randomly in different aspects of an image, applying shifts, zooms, rotations, brightness changes and even shear effects on the same input image and feeding it to the CNN. In order to speed up the learning process and work with well-known solutions, we opted for a technique known as transfer learning, where from a pre-trained model and pre-trained weights we use train our new model on the new classification task, serving as a good starting point. The models used were trained on the *ImageNet* dataset and were used as feature extractors, where we ignored the output (top) layer and added a layer depending on the problem. An image size of 224x244 pixels was maintained in order to take advantage of the pre-tained weights. Regarding the optimizer we did experiments with either *Adam* and *RMSProp* and we opted for the last one as converged faster for our problem. Due to the single class classification nature of the problem the last layer was a *Softmax* one with the number of nodes equal to the number of classes we want to predict, thus giving us the probability distribution on the classes and the loss function used was *Categorical Crossentropy*. Regarding metrics, several were keep track during the different phases (train, validation and test), from accuracy, precision, recall, Area Under the Curve (AUC) of Receiver Operating Characteristic curve (ROC), and (AUC) of precision-recall curve (PR). These metrics were used in the following models of this section. During the training *Dropout* layers and *Early Stop* depending on the AUC-PR of the validation data was used in order to prevent overfitting.

Several approaches were made, the first one, a single classifier model was responsible to classify an image into a class. To handle imbalanced data, we did a random over-sample of the less represented features, while maintaining their relative order of frequency. Moreover different class weights were passed to the training of the model in order to give higher loss to classes that are less represented in the dataset. There were made experiments with different networks, starting from the simple VGG16, but better results were achieved using a ResNet50 and InceptionV3, the last giving the best results. An fully connected layer was added to the end of the base-model and the end model was trained on the train dataset.

Model	Accuracy	F1 micro	F1 weighted
50 classes	68%	68.7%	66.3%

TABLE II: 50 classes model metrics

The metrics evaluated with this model seems rather satisfying, however we must remember the highly imbalanced data and a deeper inspection with resource to the confusion

matrix and classification report the model is ignoring the low representation data and focusing on the two most represented classes.

As second approach, due to the highly imbalanced data we decided to process and split the data into two and train two distinct models on each dataset. The first model was responsible to distinguish between the two more frequent classes (51 and 13) and the remaining of the classes (all classified as -1), whereas the second model was responsible to classify the images where the first model classified as being one of the remaining, summing up to 48. Regarding the first model, there were three classes on which the model was trained, given that their relative frequencies were fairly similar there was no need for any over-sample or the class weights approach, however for the second model, even though the relative frequencies were smaller than in the original dataset there was still some considerable difference, so class weights was added in the training process of this model and over sample was also used. In this two models we used an InceptionV3 networks. At the end of each model we added a Global Average Polling layer, which allowed to reduce the total number of parameters in the model, useful given the dataset being not huge enough, followed by a small fully connected layer. In this approach we did a fine-tuning of the models in order to take the best out of the pre-trained weights. At first we froze the base-model layers and did a training for a few epochs until the model stabilized, so that the error of the random weights would not be great enough to waste the pre-trained weight's knowledge. Then we would unfroze the base-model. In the 3 classes classification we unfroze the base-model all at once and trained, however in the 48 class model we got better results using a two-phase unfreezing, by allowing training in near half of the base model (from the end), train, and only then unfroze all the base-model, followed by a train. This way we were able to keep some of the knowledge from the *ImageNet* dataset which was useful for unseen images. We used a train-validation-test split of 68-17-15 for the both datasets. In the first model we were able to get an F1-Score of about 77,3% in the test dataset and the following metrics.

Model	Accuracy	F1 micro	F1 weighted
3 classes	77.4%	77.3%	76%

TABLE III: 3 classes model metrics

Regarding the second model, using the 3-phase training we were able to get an F1-Score of about 66% in the test dataset for this model and a the following metrics.

Model	Accuracy	F1 micro	F1 weighted
48 classes	65.5%	66%	63.3%

TABLE IV: 48 classes model metrics

Given the nature of the data we got two models that can better predict on the iMet-2019 dataset for the given classes. This first model could distinguish between the class 13 and 51 and the remaining with a fairly good accuracy, however it is not surprisingly high, maybe due to the existence of noise in the dataset and that the remaining of the classes are very diverse, so there is a lot of variance in that sense. The second model, given the disparity of frequencies and the low representation, could actually get a remarkable result in distinguishing among 48 classes, something that the previous approach hasn't been able to do, so in this aspect we considered that the two-models approach got a better and more significant representation of the information in the data than the one-model approach. With more data available for training and with a more balanced dataset we believe that the results could be better, however, many times, the models can only be as good as our data, so in this sense, making a pre-processing of data as done here helped us achieve these results. This being said, due to the low frequency of some classes it is not possible to have a clear representation of that data, specially on Neural Networks approach which need a lot of data for training their parameters, so the model is limited by the dataset.

C. Feature Extractor with Classical Classification Model

The approach of not using a deep learning model to classify the images was only done on a part of the dataset, more precisely on the images whose label was not 13 neither 51 - the more frequent classes. This decision was based on the fact that when having fewer input data but a huge number of features using a classical classification model can provide better results than Deep learning. For Deep learning, a lot of data with lots of features is required to prevent overfitting and to build a model that generalizes the data well.

For those 2 classes, the pre-trained model allied with a sufficient amount of data resulted in a network that could distinguish those types of images from the rest fairly well. For the other 48 classes, we decided to try to use a classical machine learning classification technique that typically does not need a lot of data to be trained. For the feature extraction, we used the feature map that resulted from VGG-16 and InceptionV3 models pre-trained on the *ImageNet* dataset and we decided to freeze the layers of the model since it already has the needed filters for the feature extraction. For the classification part, since we did not know which classifier might work the best, we decided to try with a few of them using different parameters.

Moreover, data augmentation was done according to the number of samples of each classe (the more samples available, the fewer the images created). After this, even though the data is still not balanced, there were introduced perturbations in the input to help in the regularization and, in this case, also helped to reduce the discrepancy in the number of samples of each class. Nevertheless, when training the model, a parameter regarding the weight of the classes is passed as *balanced* and, in the end, when measuring the performance metrics we looked for the F1 and accuracy (not as important) metrics using the micro (which is recommended when we have unbalanced classes) and we were able to achieve almost 70% of F1 score and more than 60% accuracy for the 48 classes.

Model	Accuracy	F1 micro	F1 weighted
RandomForest	54.2%	54.1%	51.3%
SVM OneVsRest	68.6%	68.3%	66.04%
SVM OneVsOne	67.7%	67.7%	66.1%
Logistic Regression	59.9%	59%	58.4%

TABLE V: Metrics using InceptionV3 as a feature extractor with data augmentation and images with 299x299x3

Model	Accuracy	F1 micro	F1 weighted
RandomForest	60.5%	60.4%	57.5%
SVM OneVsRest	66.38%	66.38%	63.4%
SVM OneVsOne	64.9%	65%	63.3%
Logistic Regression	60.9%	60.9%	58.75%

TABLE VI: Metrics using VGG-16 as a feature extractor with data augmentation and images with 224x224x3

The metrics are quite promising for the amount of data we have. Even though this is not displayed in the report explicitly, the experiment done was also run not using data augmentation and the results were at least 5% worse. Comparing them with BoVW, they are fairly better and are slightly better than using fully connected layers. This has most likely to do with the amount of data not being quite enough to train so many parameters for these types of models.

V. Multilabel Classification

Through the analysis of the results obtained for the multiclass problem, we decided that many aspects of the approach could be brought over to the multilabel problem. The use of the *ImageDataGenerator* class allowed for data augmentation, although in a more broad way, as there was no selective augmentation of less common sections of the dataset, and made use of rescaling, rotation, width shift, height shift, zoom, and horizontal flip as possible modifiers. *ResNet50* and *InceptionV3* were used in the training as they were models that achieved greater results in the multiclass classification. The models were compiled with a densely connected layer of 1024 units and had *ReLU* as the activation function, defined by its low computational cost and fast convergence. A value of 0.5 was used on a *dropout* layer to prevent overfitting during the process of training. Lastly, a densely connected layer was used as the output layer, using 72 units with the larger dataset, and 28 units for the smaller one, corresponding to the number of different labels present in the respective datasets. In this layer the *Sigmoid* activation function was used for its resulting outputs being easily used as percentage values for each label.

The datasets not only present the corresponding image id and set of labels, but also a set of binary values related to the correspondence of the image to a given label. This data is relevant when using *binary_crossentropy* as a loss function, due to the fact that the labels being able to be reduced to simpler classifications (for example: given a label A, there is classification of whether the image corresponds to A or not A).

The training from five epochs results in the following metrics (the f2 score was calculated with the recall and precision values):

Model	Accuracy	Precision	F2
ResNet50	45.8%	68.3%	25%
InceptionV3	47.7%	60.3%	32%

TABLE VII: Validation results from the multilabel models

From the results, it's noticeable that there isn't much of a difference between *ResNet50* and *InceptionV3* when it comes to accuracy. On the other hand, there are larger variations between the precision and F2 score values, being *ResNet50* the one with the higher precision, and *InceptionV3* with the higher F2 score. From a general standpoint, considering the limited time given and that multilabel classification is a complex problem to solve, the results achieved could be considered a success, even if not quite usable in a practical sense, having only around 45% accuracy for both models used.

VI. Conclusions

In this report we were able to explore different methodologies and techniques used in image classification, as well as experiments with the state of the art architectures. Due to the nature of this problem we were faced with several difficulties, from data imbalance to different image shapes, which allowed to explore and broaden our knowledge on this field. Convolutional Neural Networks are the standard approach to image processing applications, however we got surprising results when applying a Support Vector Machine to our dataset, which is very interesting and further work could be done in that direction. Also, noise removal could also be implemented as future work, by training an initial model and removing entries for which the error is considerably high, and then use that data as training data, extracting outliers from the dataset. This are some ideas regarding future work, however overall the work developed showed how Convolutional Neural Networks, either being used as transfer learning model either as feature extractor, play a big role in image application and processing, being capable of real context application.

References

[1] https://keras.io/guides/transfer_learning/, Keras transfer learning guide, Chollet, F. (2020, May 12)

[2] https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/, Transfer learning in keras with computer vision models, Brownlee, J. (2020, August 18)

[3] https://keras.io/api/applications/, Keras applications, Keras

[4] https://www.learndatasci.com/tutorials/hands-on-transfer-learning-keras/, Hands-on transfer learning with keras and the VGG16 model, McDermott, J.

[5] https://machinelearningmastery.com/how-to-implement-major-architecture-innovations-for-convolutional-neural-networks/, How to develop VGG, inception and ResNet modules from scratch in keras, Brownlee, J. (2019, July 5)

[6] https://www.tensorflow.org/tutorials/structured_data/imbalanced_data, Classification on imbalanced data, TensorFlow (2021, May 25)

[7] https://keras.io/api/preprocessing/image/, Image data preprocessing, Keras

[8] https://www.machinecurve.com/index.php/2020/11/10/working-with-imbalanced-datasets-with-tensorflow-and-keras/, Working with Imbalanced Datasets with TensorFlow 2.0 and Keras, Versloot, C. (2021, January 20)

[9] A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK. Tareen, Shaharyar Ahmed Khan Saleem, Zahra (2018), 10.1109/ICOMET.2018.8346440.

[10] Sampling Strategies for BagofFeatures Image Classification. Nowak E., Jurie F. Triggs B. (2016), Proceedings of the European Conference on Computer Vision, pp. 490-503

[11] Imagenet classification with deep convolutional neural networks, Krizhevsky A., Sutskever I. Hinton G. E. (2012), Advances in neural information processing systems, pp. 1097-1105
[12] You only look once: Unified, real-time object detection. Redmon J., Santosh D., Girshick R. Farhadi A. (2016), Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779-788
[13] LearningDeep Features for Discriminative Localization. Zhou B., Khosla A., Lapedriza A., Oliva A. Torralba A. (2016), Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2921-2929
[14] https://towardsdatascience.com/4-ways-to-improve-class-imbalance-for-image-data-9adec8f390f1, 4 Ways to Improve Class Imbalance for Image Data, Potyraj E. (2021, March 8)
[15] https://scikit-learn.org/stable/, Machine Learning in Python, scikit-learn
[16] Comparative study of k-means and mini batch k-means clustering algorithms in android malware detection using network traffic analysis. Feizollah A., Anuar N. B., Salleh R. Amalina F. (2014), International Symposium on Biometrics and Security Technologies (ISBAST), pp. 193-197, 10.1109/ISBAST.2014.7013120.

APPENDIX
VII. MORE RESULTS

```
test_generator.reset()
evaluate = model2.evaluate(test_generator)
evaluate
```

```
208/208 [==============================] - 91s 437ms/step - loss: 2.8431 - acc: 0.6880 - precision: 0.6978 - recall: 0.6765 - auc: 0.9309 - prc: 0.6805
```

Fig. 5: 50 classes model metrics

```
test_generator.reset()
evaluate = model1.evaluate(test_generator)
evaluate
```

```
208/208 [==============================] - 71s 341ms/step - loss: 0.7874 - acc: 0.7740 - precision: 0.7967 - recall: 0.7508 - auc: 0.9159 - prc: 0.8553
```

Fig. 6: 3 classes model metrics

```
test_generator.reset()
evaluate = model2.evaluate(test_generator)
evaluate
```

```
52/52 [==============================] - 12s 221ms/step - loss: 3.4627 - acc: 0.6553 - precision: 0.6734 - recall: 0.6480 - auc: 0.8895 - prc: 0.6229
```

Fig. 7: 48 classes model metrics

```
596/596 [==============================] - 195s 327ms/step - loss: 0.0550 - acc: 0.4520 - precision: 0.7697
- recall: 0.2372 - auc: 0.9439 - prc: 0.4798 - accuracy: 0.4520 - val_loss: 0.0711 - val_acc: 0.4581 - val_
precision: 0.6825 - val_recall: 0.2714 - val_auc: 0.9007 - val_prc: 0.4640 - val_accuracy: 0.4581
```

Fig. 8: Multilabel validation results for the ResNet50 model

```
596/596 [==============================] - 192s 322ms/step - loss: 0.0451 - acc: 0.6030 - precision: 0.8261 - recall: 0.4262
- auc: 0.9600 - prc: 0.6607 - accuracy: 0.6030 - val_loss: 0.1359 - val_acc: 0.4773 - val_precision: 0.6029 - val_recall: 0.
3897 - val_auc: 0.8512 - val_prc: 0.4343 - val_accuracy: 0.4773
```

Fig. 9: Multilabel validation results for the InceptionV3 model