Multi-label image classification

LDA-T308 Introduction to deep learning

Mariia Shpir, Jani Kuurasuo

2023

1 Introduction

Multi-label image classification is a one of the computer vision tasks, which involves predicting multiple labels for an image. In this project, we aim to build a deep learning model that can accurately classify diverse objects in an image. We will use a dataset provided by the University of Helsinki, which consists of images and their associated labels. The dataset is not exclusive, meaning that an image can have multiple labels.

The goal of this project is to explore the effectiveness of deep learning models for multi-label image classification tasks. We will train Residual Neural Network and Visual Transformers models that can accurately predict multiple labels for an image. Our report will detail the preprocessing steps, model architecture, training process, and evaluation metrics used in the project.

2 Data

2.1 Data preprocessing

We examined the data by plotting the distribution of different labels in the data set. Some labels were underrepresented, as shown in Figure 1. We used 2 different techniques to counteract this: oversampling method and weighted loss.

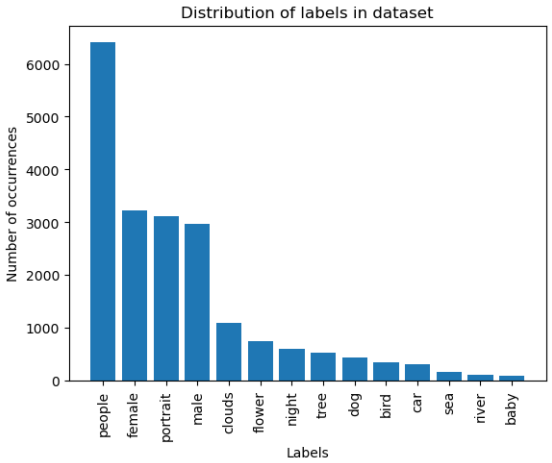

Figure 1: The y-axis shows the number of occurrences of each label in the data set.

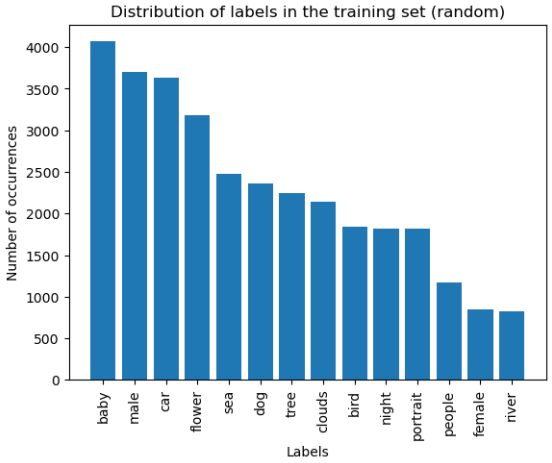

Figure 2: Label distribution for a balanced training set.

Data oversampling

We implemented an existing sampler that was available on GitHub: (Memari). It samples each class with equal probability, sampling more from under-represented classes. The label distribution for a balanced training set is shown in Figure 2.

However, since multiple labels could be present in a single image, the resulting distribution was not completely balanced. As a result, labels that frequently occurred together were somewhat over-represented. We attempted to solve this issue by exploring alternative approaches, but then end up using MultiBalancedSampler due to its effectiveness.

Weighted loss

According to the PyTorch 2.0 documentation of BceWithLogitsLoss: ”It’s possible to trade off recall and precision by adding weights to positive examples. ... For example, if a dataset contains 100 positive and 300 negative examples of a single class, then pos_weight for the class should be

Metric	Training set		Validation set	
	Primary	Transformed	Primary	Transformed
Accuracy	0.92	**0.93**	0.92	**0.93**
Loss	0.2	0.18	0.18	0.19

Table 1: The results of the CNN model performed with and without data augmentation methods.

equal to $300/100 = 3$. The loss would act as if the dataset contains $3 * 100 = 300$ positive examples."

This approach was implemented in get_class_weights() function presented in the code snippet. As can be seen from Figure 3, the weights distribution corresponds to label distribution.

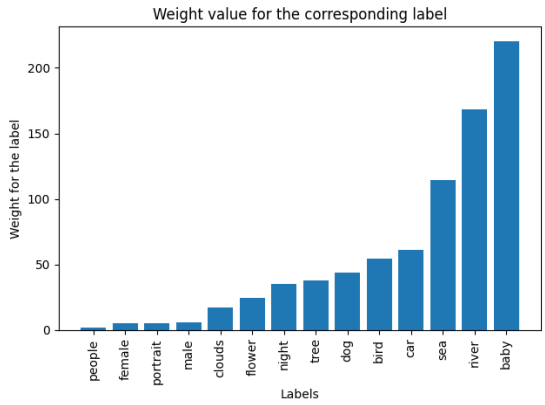

Figure 3: Label distribution for a balanced training set.

3 Regularization

In our experiments, we observed that our models were performing significantly better on the training set than on the validation set, leading us to explore various methods for regularization.

3.1 Data augmentation & Normalization

Before the training, we normalized the images using the transforms.Normalize provided in the PyTorch library. For the ViT, ResNet152 and DenseNet161 model, we resized the given images to the size required by the corresponding pre-trained model. In addition, to prepare data for convolutional models, we used various data augmentation techniques: RandomHorizontalFlip, RandomVerticalFlip, ColorJitter with the results presented in Table 1.

3.2 Weight decay

The L2-norm is a regularization method that involves adding a penalty to the loss function of the network, which pushes the weights toward zero. This can reduce overfitting, making the model less sensitive to noise in the training data and more generalizable to unseen data.

We tested L2 regularization on ResNet152 with varying values for the decay hyper-parameter. However, any amount of weight decay increased our error drastically, indicating that L2 regularization was not an effective method for improving the performance of our model.

3.3 Dropout

Dropout is a regularization technique where the value of some neurons is randomly set to zero during an iteration. As a result, the network learns to not rely too heavily on one specific feature, thus reducing overfitting.

To apply dropout in our models, we added layers with a hyperparameter of 0.5, which denotes the fraction of neurons in the network to be dropped out. Our experiments showed that the addition of dropout layers increased the micro-averaged f1-score by 1% and the macro-averaged f1-score by 2%, indicating a significant improvement in performance.

4 Optimization

4.1 Adam

For our optimization algorithm, we chose the Adaptive Moment Estimation (Adam), which is very commonly used for training neural networks. It is a variation of the SGD algorithm, combining features of AdaGrad and RMSProp. It functions by computing adaptive learning rates for each parameter based on the first and second moments of the gradients.

4.2 Learning rate scheduler

Learning rate schedulers are used in optimization to adjust the learning rate during the training process. We used LinearLR, CosineAnnealingLR from torch.optim in PyTorch. Adding LinearLR improved the macro-averaged f1-score by 2%, while using CosineAnnealingLR decreased both the micro and macro-averaged f1-scores.

LinearLR is given a factor, in our case $\frac{1}{3}$, and the number of iterations. At the first epoch, the learning rate is equal to this factor multiplied

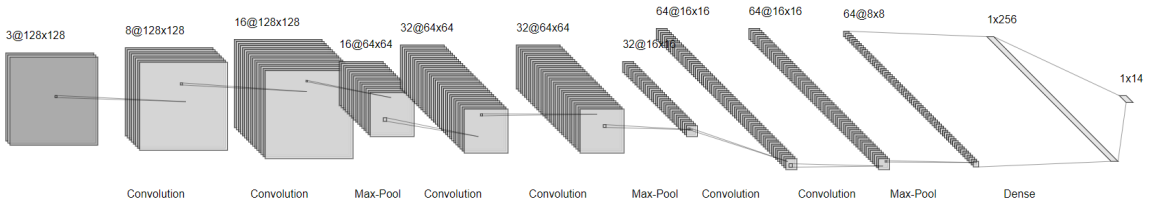

Figure 4: An architecture of the trained CNN model.

by the learning rate given to the optimizer. As the number of epochs gets closer to the given number of iterations, this factor gets closer to 1, so that at the 10th iteration, the learning rate is the original learning rate given to the optimizer.

4.3 Batch normalization

Batch normalization is a optimization technique that minimizes internal covariate shift, which can require small learning rates and careful parameterization. In batch normalization, mean and variance of inputs are calculated for each layer for a batch of training data, which are then used as a shift and a scale parameter to normalize the inputs: $y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$, where $\gamma^{(k)} = \sqrt{\text{Var}[x^{(x)}]}$, and $\beta^{(k)} = E[x^{(x)}]$ (Ioffe and Szegedy, 2015).

For reference, we tested ResNet152 with and without batch normalization in the added fully connected layers, and found that removing it decreases our validation F1-score by almost half.

5 Loss function & Metrics

In this project, we use the binary cross-entropy loss function, which is widely used for multi-label classification tasks. The binary cross-entropy loss calculates the error between the predicted and actual labels for each class and averages the errors across all classes. We imported the BCEWithLogitsLoss function from the PyTorch library.

In terms of evaluation metrics, we used several standard metrics for multi-label classification task imported from sklearn.metrics library: accuracy, both micro and macro averaged precision, recall, and F1-score. For a more accurate results, we also used multi-label confusion matrix.

Metric		Unbalanced	Balanced	Weighted
Accuracy		0.93	0.92	**0.94**
Loss		0.18	0.19	0.29
Micro	F1 score	0.19	0.07	**0.69**
	Precision	0.68	0.72	0.55
	Recall	0.11	0.03	0.91
Macro	F1 score	0.04	0.01	0.49
	Precision	0.13	0.07	0.42
	Recall	0.03	0.009	0.65

Table 2: The results of the CNN model performed on the validation set.

6 Models

6.1 CNN

Convolutional Neural Networks (CNNs) are effective in diverse computer vision tasks, including multi-label image classification. We constructed a model with the architecture presented in Figure 4 and trained it using unbalanced, balanced, and weighted data with results in Table 2.

We suggest that the low performance of the model can be performed due to several factors, including the limited diversity in the dataset, the inability to capture global patterns and features, and the difficulty in handling complex interactions between the labels.

6.2 ResNet & DenseNet

”Residual Network”, or ResNet, is a deep neural network architecture that was introduced in a research article by He et al. in 2015 (He et al., 2015). Its architecture is comprised of residual blocks containing convolutional layers, and have residual connections that allow the block to transfer information to deeper layers, skipping any unwanted intermediate blocks.

After experimenting with CNNs, we decided to try

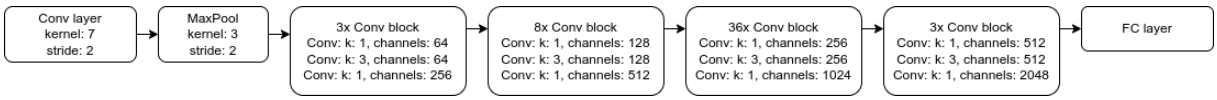

Figure 5: An architecture of ResNet152.

Metric	Training set		Validation set	
	RN152	DN161	RN152	DN161
F1 (micro)	**0.67**	0.60	**0.72**	0.65
F1 (macro)	0.42	0.35	0.44	0.39

Table 3: Comparison of ResNet152 and DenseNet161 results.

transfer learning using pretrained models, specifically ResNet50 and Resnet121, DenseNet121 and DenseNet161. It is worth clarifying that the models ResNet50 and DenseNet121 showed worse results than their more complex versions, and therefore are not considered further.

We used pretrained and default IMAGENET1K_V1 weights for ResNet152 and DenseNet161 from PyTorch library, and added fully connected layers on top shown in Figure 6, with the input dimension of the first linear layer being 2048x2048 for ResNet152 and 2208x2208 for DenseNet161. Table 3 shows a comparison between the micro and macro-averaged F1-scores for both models, ResNet152 produced better results. The final results of the Resnet152 model are presented in the Table 4 for further comparison.

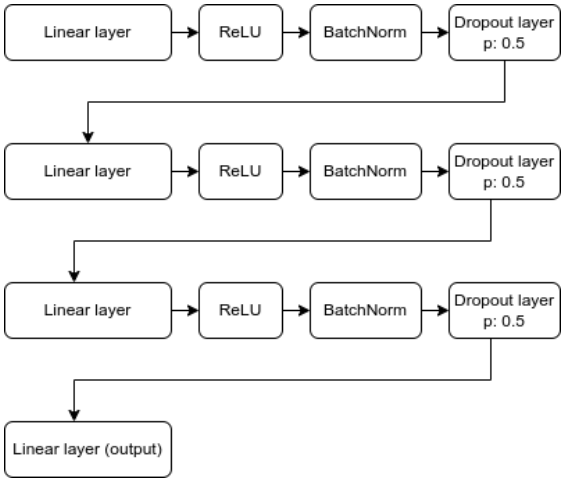

Figure 6: Architecture of the added fully connected layer for pretrained models.

Metric		Training set		Validation set	
		U	B	U	B
Accuracy		0.95	0.96	**0.96**	**0.96**
Loss		0.12	0.11	0.09	0.1
Micro	F1 score	0.64	0.67	**0.71**	**0.71**
	Precision	0.75	0.77	0.82	0.8
	Recall	0.57	0.61	0.64	0.65
Macro	F1 score	0.38	0.42	0.43	0.44
	Precision	0.45	0.39	0.49	0.5
	Recall	0.36	0.48	0.4	0.42

Table 4: The results of the ResNet152 model performed on the validation set with U - unbalanced and B - balanced.

Metric		Training set			Validation set		
		U	B	W	U	B	W
Accuracy		0.98	0.98	0.94	**0.97**	**0.97**	0.95
Loss		0.06	0.06	0.2	0.07	0.07	0.33
Micro	F1 score	0.84	0.83	0.72	**0.82**	**0.82**	0.74
	Precision	0.86	0.86	0.59	0.83	0.85	0.63
	Recall	0.82	0.82	0.94	0.81	0.79	0.92
Macro	F1 score	0.54	0.55	0.52	**0.53**	**0.54**	0.53
	Precision	0.57	0.57	0.45	0.55	0.58	0.47
	Recall	0.54	0.55	0.68	0.53	0.54	0.67

Table 5: The results of the ViT model performed on the validation set with U - unbalanced, B - balanced and W - weighted data.

6.3 ViT

ViT (Dosovitskiy et al., 2021) processes images by dividing them into smaller patches and using transformer blocks. This allows to capture the global context of the image and understand its overall meaning. ViT has achieved competitive performance compared to convolutional neural networks (CNNs). This approach has the potential to be a powerful tool for multi-label classification tasks, where understanding the overall meaning of an image is important.

We implemented ViTForImageClassification model from Hugging Face library, which was pretrained on a large collection of images in a supervised fashion, namely ImageNet-21k, at a resolution of 224x224 pixels. We also added fully connected layers on top shown in Figure 6. The results presented in Table 5.

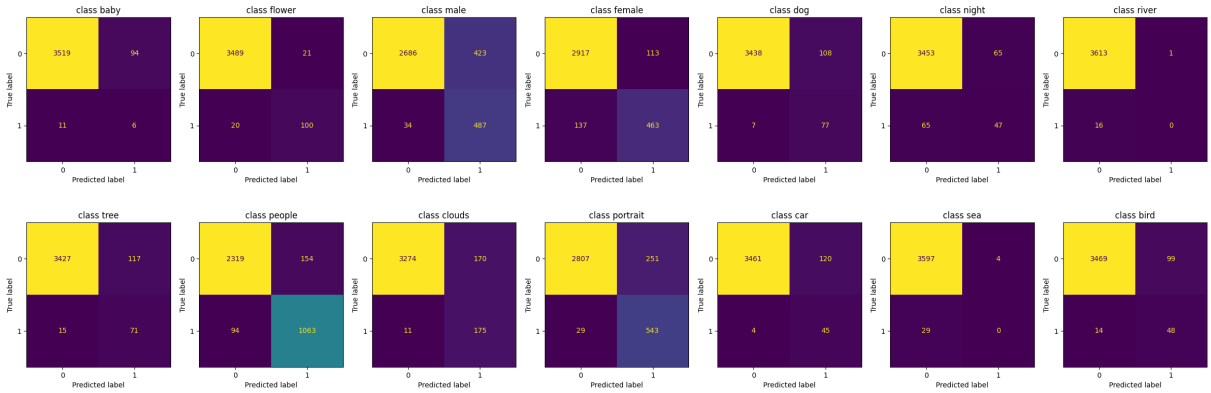

Figure 7: Confusion matrices for each presented label.

The attention mechanism allows to learn more global features and long-range dependencies and it is highly useful in multi-label classification tasks where identifying the relationships between different objects in an image is important. In addition, such a model is faster to train and requires less computational cost since it has fewer parameters. To summarize, the ViT model shows better accuracy than other tested approaches with the same result on both unbalanced, balanced and weighted data.

7 Error analysis & Results

The entire error analysis is based on the results of the ViT model, trained with both balanced and weighted data. This approach increased macro F1-score up to 65 %. More detailed results presented in the Table 6.

Figure 8: Misclassified examples of the class 'portrait'.

Figure 7 shows a visualization of the multi-label confusion matrix from the sklearn.metrics. Based on the results, we can conclude that the model has difficulty classifying 'portrait', 'people', 'female' and 'male' classes.

Metric	Training set	Validation set
Accuracy	0.96	0.96
Loss	0.16	0.55
Micro F1 score	0.79	0.75
Macro F1 score	0.66	0.65

Table 6: The results of the final ViT model with balanced and weighted data.

The model's classification of example 1 in Figure 8 as non-portrait may have been due to the positioning of the face and low quality. In contrast, example 3 lacks a discernible face but may have been classified as a portrait based on the outlines.

Figure 9: Misclassified examples of the class 'people'.

In Figure 9 the examples 2-4 were falsely classified as non-people. This may be due to the low quality of the images. It is likely that if the photos were of higher quality, the model would have identified them as people.

Figures 10 and 11 show both image quality issues and the limited scope of the dataset. We suggest that to improve model's accuracy, it is worth to improve the dataset by including underrepresented classes, improving photo quality, and reconsidering photo labels. In addition, the solution would also

Figure 10: Misclassified examples of the class 'female'.

Figure 11: Misclassified examples of the class 'male'.

be to increase the complexity of the model, which requires more computational power and better regularization.

References

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

Issa Memari. Pytorch multilabel balanced sampler.