# Stay Focused: Our solution to AI@UNICT 2023 - Image classification with distribution shift

**Orazio Pontorno** OPONTORNO@GMAIL.COM

## 1. Introduction

The aim of the challenge was to build a model capable of classifying eight types/classes of objects in different images. These classes were as follows: '*plug*', '*mobile phone*', '*scissor*', '*lamp*', '*pepsi*', '*sunglasses*', '*ball*' and '*cup*'.

The dataset containing the images is available at the following link.

The images are already divided into train and test images and, in addition, for each one we find a file in *.csv* format containing for each image the coordinates of the buonded box enclosing the object, which will be useful for focusing the network only in the region of the pixels that draw the object.

Let us focus our attention on the train dataset. As already mentioned, the images differ in eight object classes. For seven of them, in particular for all classes except the class '*plug*', the objects occur in different backgrounds. For images with class *plug*, all images have the same background, which is not shared with any other class.

Figure 1: Images of train dataset.

This leads to each classifier network having a strong bias against this class. Moreover, in the images of the test dataset, all objects have the same background, the same as the *plug* class. The real challenge therefore lies in eliminating this bias in the decision-making process of the model.

Figure 2: Images of test dataset.

## 2. Our Solutions

Three different approaches have been tried to overcome this problem:

- The first approach consisted in using a simple Convolutional Neural Network (CNN), trying to make the network focus on the object in question through the use of Data Augmentation techniques.
- The second approach, very similar to the first, consisted in using a state-of-art CNN and manually removing the bias of the *plug* class. In addition, some Data Augmentation transformations were applied to the images.
- The third and final approach consists of using a state-of-art Object Detector network.

Unlike the last two, the first approach did not yield great results. In fact, despite numerous attempts to modify the network's hyperparameters and the different combinations of Data Augmentation transformations used, the maximum accuracy value achieved was **33%**. Much better performance was achieved with the second and third approaches. Average accuracy values of approximately **96%** in the second approach and **95%** in the third approach were obtained. In both approaches, the same data augmentation transformations were performed and the coordinates of the bounded boxes were used, albeit in a completely different way: in fact, while in the second approach they were useful for removing the background in the offending class, in the third approach they were passed to the object detector network in charge not only of classifying the object class, but also of predicting the coordinates of the bounded box that delimits it.

The models used in the second approach were the *ResNet101* and the much lighter (in terms of number of parameters) *GoogleNet*, while in the third approach the *Faster R-CNN* was used.

The CNN networks used in the second approach are much less expensive in terms of complexity and computational time than the *Faster R-CNN* object detector, but despite these, the former obtained performances comparable to those obtained by the latter. For this reason, data preprocessing and augmentation processes, training and the results obtained in the second approach will be presented in this report.

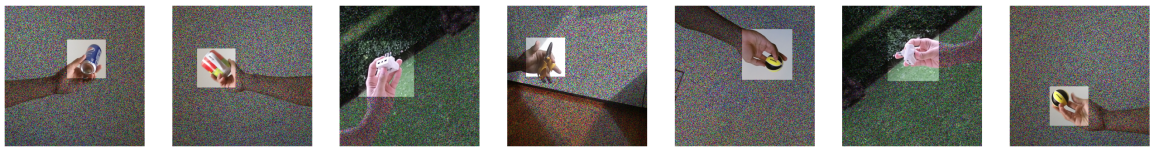
## 3. Model-building phase

We now enter the model building phase. In this section, we will expose the Data preprocessing and Data Augmentation phases, then move on to the description and explanation of the model architecture, and then finish by exposing the model training procedure.

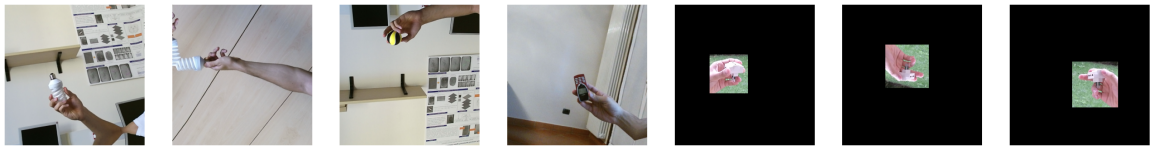
As mentioned in the previous section, the models used using the *fine tuning* technique were two: the *ResNet101* and the *GoogleNet*. But in this report we will only focus on the former, as it led to higher accuracy values.

### 3.1. Data Preprocessing and Augmentation

Before performing Data Augmentation techniques, the images underwent several 'handmade transformations'. These transformations consisted of ruining the images, containing objects belonging to one or more classes, in areas where the object to be classified was not present. To do this, use was made of the bounded boxes provided and the application of Hademard's product between tensors. In particular, a tensor was created with the same dimensions as the image tensors filled with random or chosen values (depending on the type of transformation chosen) except in the area swept by the box, where it was filled with one. Several datasets were created for each type of transformation, which were subsequently joined in different combinations, in pairs or triples, to train the model. Below we see three examples of datasets generated using these transformations.



(*a*) Dataset with 'blur' transformation to all classes



(*b*) Dataset with 'black' transformation only to class 'plug'

(*c*) Dataset with 'random' transformation only to class 'plug'

Figure 3: Datasets generated using different transformations

In these dataset examples you can see the three types of transformations applied to the dataset. In figure 3.a we see a generated dataset applied to each image the transformation ' *blur*', consisting in the ruin of the background; While in figure 3.b and in figure 3.c, only the images containing objects of the class *plug* making the background black and a random color, respectively. The choice to generate these last two types of datasets comes from an empirical improvement in the performance of the model. In fact, Precisely the dataset resulting from the concatenation of these last two datasets has been used to train the model.

It is necessary to specify that, in order to avoid the risk of having images not belonging to the class *plug* duplicate, different transformations of Data Augmentation have been applied. To do this we used the library **Albumentation**, which allows you to make transformations simultaneously to both images and bounded boxes inside. The applied transformations were the following: *HorizontalFlip()* with probability equals to 0.5, *VerticalFlip()* with probability equals to 0.5, *ShiftScaleRotate()* with probability, scale limit and rotate limit respectively equal to 0.5, 0.2 and 45. In addition, Data Augmentation is crucial to improve the accuracy and generalization capability of deep learning models. By providing models with a larger and more diverse dataset, they can learn to recognize different models and features.

### 3.2. Model Description

*ResNet101* is a convolutional neural network architecture that was proposed in the paper "Deep Residual Learning for Image Recognition" by Kaiming He et al. in 2016. It is an extension of the *ResNet50* architecture, and has 101 layers. *ResNet101* is a very deep and powerful architecture that has achieved state-of-the-art performance on many image recognition tasks.

The main feature of the Net is the so-called **residual block**. In traditional neural networks, information is passed through the layers in a sequential manner, with each layer transforming the

input from the previous layer. However, as the number of layers increases, the vanishing gradient problem occurs, making it difficult to train the network effectively.

ResNet solves this problem by introducing residual connections between layers. The residual block contains two paths: the "shortcut" path, which bypasses one or more layers and connects the input directly to the output, and the "main" path, which performs the usual transformations on the input. The output of the shortcut path is then added element-wise to the output of the main path. This allows the gradient to flow directly through the shortcut path, making it easier to train very deep networks.

The residual block can be expressed mathematically as:

$$y = \mathcal{F}(x, \{W_i\}) + x$$

where $\mathbf{x}$ is the input to the block, $\mathcal{F}$ is the residual function consisting of a sequence of layers, $W_i$ are the weights of the layers, and $\mathbf{y}$ is the output of the block, which is the sum of the output of $\mathcal{F}$ and the input $\mathbf{x}$.

*ResNet101* uses residual blocks, consisting of 3 convolutional layers, to enable the training of very deep neural networks.

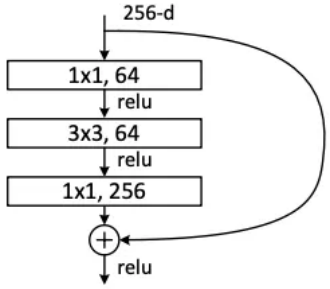



Figure 4: Residual blocks in ResNet101

The general structure of *ResNet101* can be broken down into 5 stages:

- Convolutional layer: A 7x7 convolutional layer with stride 2 that performs downsampling of the input image.
- Max pooling: A 3x3 max pooling layer with stride 2 that further downsamples the input image.
- Stage 1: A sequence of residual blocks with 64 filters. This stage has 3 residual blocks.
- Stage 2: A sequence of residual blocks with 128 filters. This stage has 4 residual blocks.
- Stage 3: A sequence of residual blocks with 256 filters. This stage has 23 residual blocks.
- Stage 4: A sequence of residual blocks with 512 filters. This stage has 3 residual blocks.

At the end of the last stage, there is a global average pooling layer, followed by a fully connected layer with 1000 units (for the ImageNet classification task) and a softmax activation function to produce the final class probabilities. Obviously, the last fully connected layer was adapted to our problem, so it underwent a change in the number of output neurons from 1000 to 8.

### 3.3. Training procedure

The training procedure in deep learning is the process of training a neural network model to learn from data and make accurate predictions. It involves several key steps that are repeated iteratively to optimise the performance of the model. This is referred to as the *Training loop*, i.e. iterating the training dataset in batches. For each batch, perform the following steps:

1. Forward pass: Feed the batch of input data through the network to obtain the predictions.
2. Loss calculation: Compares the model predictions with the actual labels and calculates the loss using the selected loss function.
3. Backpropagation: Calculates loss gradients with respect to model parameters. This is done by propagating the backward error gradients through the network, layer by layer.
4. Parameter update: Use the optimization algorithm to update the model parameters based on the calculated gradients. This step adjusts the weights and biases of the network, with the goal of reducing loss.

The final performance of the model depends very much on the choice of *Hyperparameters* in the training procedure. The best configuration found to train our model with our dataset is the following:

- Optimizer: The Stochastic Gradient Descent (SGD) optimiser with a learning rate (lr) of 0.004 is used. The optimiser is initialised with the model parameters and updates them during training. The decay of the weights (w_decay) is set to 0.00001 and applies the L2 regularisation to the model weights.
- Loss Function: The loss function chosen is CrossEntropyLoss. This loss function is commonly used for multiclass classification problems.

$$\text{Cross Entropy Loss} = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} y_{nc}\log(p_{nc})$$

  where $N$ is the number of samples for each batch, $C$ is the number of classes in the classification problem, $y_{nc}$ is the ground truth label for the nth training example and cth class, and $p_{nc}$ is the predicted probability for the nth training example being classified as the cth class.
- Batch Size: The batch size is set to 8, indicating that the model will process 8 samples at a time during each forward and reverse pass. The choice of batch size may affect the training speed and memory requirements.
- Number of epochs: training will be performed for 10 epochs. An epoch represents one complete pass through the entire training data set. The number of epochs determines how many times the model will update its parameters.
- Initial Learning Rate: The initial learning rate (initial_lr) is set to 0.004. It is the step size at which the optimiser adjusts the model parameters during each update. It affects the convergence speed and accuracy of the training process.
- Weight decay: Weight decay (w_decay) is set to 0.00001 and applies L2 regularisation to the model weights. It helps prevent overfitting by adding a penalty term to the loss function based on the magnitude of the weights.

The learning curves drawn during the training procedure are shown below.

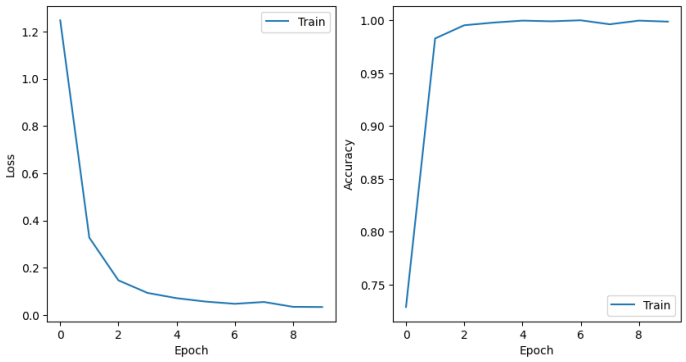



Figure 5: Loss and Accuracy curves drawn during the training procedure

As can be seen from the plots, the curves maintained an optimal trend resulting in a final loss value of approximately 0.005 and a final accuracy value close to 100%.

## 4. Experimental Results

We then come to the testing phase of the model. This is the phase in which we realise whether the model was really able to be unaffected by the background of the images in making its prediction regarding the class of the target object.

In addition to a numerical result, we made use of the *Explanability of Artificial Intelligence (XAI)* algorithm **GradCAM**, from the *camput* library. *GradCAM (Gradient-weighted Class Activation Mapping)* is a technique used in the interpretability and visualisation of deep neural networks, particularly in convolutional neural networks (CNNs). It provides insights into which regions of an input image are most relevant for the network's prediction by generating a heatmap.

In the following, we show some examples of applying the algorithm *GradCAM* to the model and see which image pattern it focused on to give its prediction.

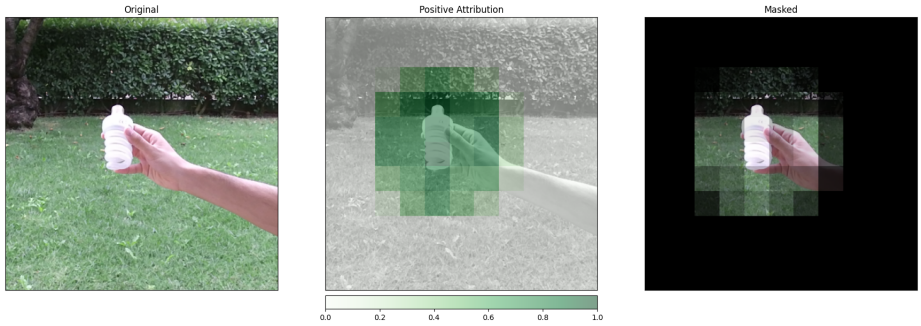



Figure 6: GradCAM on lamp testing image

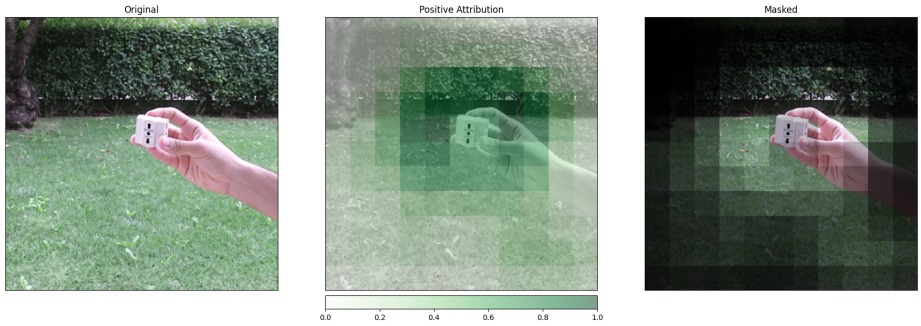



Figure 7: GradCAM on plug testing image

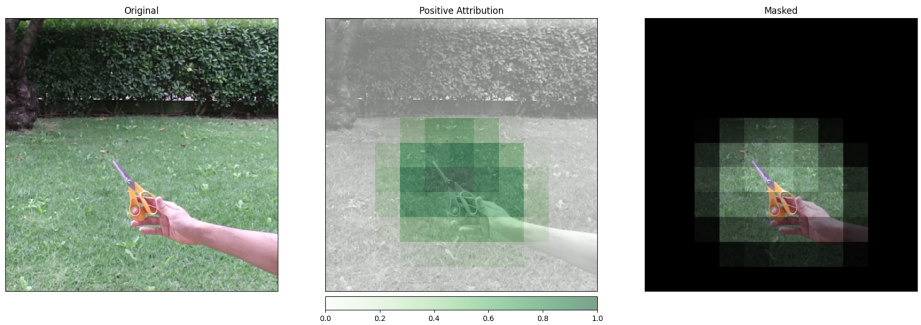



Figure 8: GradCAM on scissor testing image

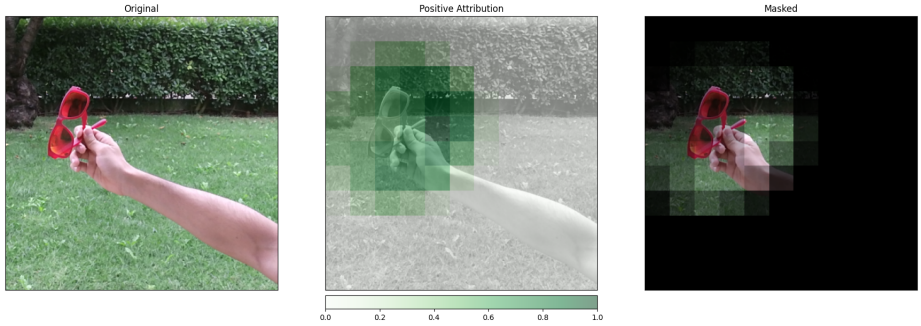



Figure 9: GradCAM on sunglasses testing image

As can be seen from the images, the model actually manages to focus on the object of the image while giving less importance to the background. From this it is therefore possible to deduce the goodness in the model. This goodness was also confirmed by numerical results, obtaining an average accuracy value of 96% (whenever the training process takes place, some random component may alter the final result), reaching a maximum value of 99%.