Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Final Project

# CNN-Based Rotation Correction

*Author:*
Matteo Breganni - 869549 - m.breganni@campus.unimib.it

January 31, 2025

**Abstract**

This project applied CNN-based methods to the task of predicting the rotation of images. Initially, the focus was on quarter-rotations (0°, 90°, 180°, and 270°), comparing hand-crafted CNNs trained from scratch, feature extraction, and fine-tuning approaches. The most effective method was then extended to handle full-range rotations.

# 1 Introduction

In recent years, convolutional neural networks (CNNs) have demonstrated exceptional performance across a variety of computer vision tasks. This project looked to take advantage of them for the task of predicting (and therefore correcting) image rotations.
In the first part, only quarter-rotations (0°, 90°, 180°, and 270°) were considered, applying different CNN-based techniques: hand-crafted CNNs that could be trained from scratch, SVM (Support Vector Machine) trained on the images features extracted from a pre-trained CNN network, and the fine-tuning of a pre-trained network.
Given the small size of the dataset, it was assumed that feature extraction or fine-tuning methods would work better; however, this turned out not to be the case.
The method that worked best was then applied to the much harder task of predicting full-range rotations, which required changes in how the input images were prepared and how the output layer was handled.

# 2 Dataset and Preparation

The dataset used in this work was manually created by sampling and filtering images from the datasets *Landscape Pictures* [1] and *Unpaired Day and Night cityview images* [2]. Only a small subset of these images were taken, and the images were selected to remove those with watermarks, borders, text of any kind, repeated images, and those that were not somewhat level.
The final dataset was made up of only 300 images: 110 from the *landscapes* dataset and 190 from the *city day images* dataset. The images are highly heterogeneous in resolution and proportion.

The dataset was divided into a training set (60%), a validation set (20%), and a test set (20%). Each image was then split into one or more square images to augment the size of the dataset and to avoid warping the images later when they will need to be given as input to the neural network models. The images are divided into one or more squares, with each square's side length equal to the smaller dimension of the original image. This method extracts the smallest number of sub-images that cover the entire original image. It is important to do this step after the dataset division; otherwise, different portions of the same image might end up in different sets.
The dataset was ultimately comprised exclusively of square images, with different resolutions (this can be easily handled by resizing the images while importing them, but leaves open the possibility of trying models with different input sizes).

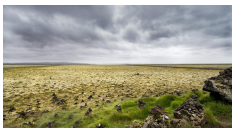

Figure 1: Image 1

Figure 2: Split Image 1

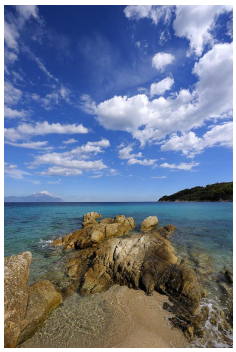

Figure 3: Image 2

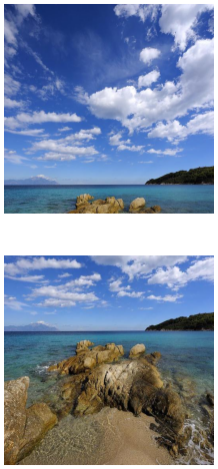

Figure 4: Split Image 2

The train set became 359 images, and the val and test sets, 116 and 117.

Lastly, the data loaders were created to import the images as the model trains to avoid exceeding the RAM limit. The data loaders take care of the preprocessing (resize, normalization, or custom processing...) and apply a random clockwise quarter-rotation. The labels were one-hot encoded.
The test set is created and saved on the disk by rotating each test image in every one of the 4 possible rotations.

# 3 Predicting Quarter-Turn Rotations

## 3.1 Hand-Made CNN Approach

The first approach was to create a CNN model and train it from scratch. The following [Table 1] shows the best model architecture that was created. Adam was used as the optimizer, with a learning rate of 0.0001. Early stopping was used. It was attempted to use L1 and L2 regularization, data augmentation through translations, changes in contrast, brightness, etc., but none of these helped the performance.

Table 1: Hand-Made Architecture Overview

| **Layer** | **Output Shape** | **Param #** |
|---|---|---|
| input | (224, 224, 3) | / |
| conv2d | (111, 111, 16) | 448 |
| batch_normalization | (111, 111, 16) | 64 |
| conv2d_1 | (55, 55, 32) | 4,640 |
| batch_normalization_1 | (55, 55, 32) | 128 |
| conv2d_2 | (27, 27, 64) | 18,496 |
| batch_normalization_2 | (27, 27, 64) | 256 |
| conv2d_3 + Dropout (40%) | (13, 13, 128) | 73,856 |
| conv2d_4 + Dropout_1 (50%) | (6, 6, 128) | 147,584 |
| flatten | (4608) | 0 |
| dense + Dropout_2 (50%) | (128) | 589,952 |
| dense_1 | (4) | 516 |

### 3.1.1 Results and Evaluation

Table 2: Hand-Made CNN's Classification Report

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
|---|---|---|---|---|
| 0 | 0.98 | 0.97 | 0.97 | 117 |
| 90 | 0.97 | 0.99 | 0.98 | 117 |
| 180 | 0.99 | 0.97 | 0.98 | 117 |
| 270 | 0.97 | 0.98 | 0.97 | 117 |
| **Accuracy** | 0.98 | | | 468 |
| **Macro Avg** | 0.98 | 0.98 | 0.98 | 468 |
| **Weighted Avg** | 0.98 | 0.98 | 0.98 | 468 |


Figure 5: Hand-Made Model Errors

### 3.1.2 Discussion

Even with the small dataset, the model's performance is extremely good. Such a high performance on model trained from scratch was not expected.
Since the mistakes made by the model were very little, it was possible to directly plot the images that were mistakenly classified [Figure 5]. Interestingly, most of the mistakes were made on the same 2 images, but with different rotations. This is a clear indicator of how the small dataset can be an issue, because since the mistakes are clustered on almost all rotations of just a couple of images, these images must have some characteristics that are not present in the training set. Having a bigger and broader training set would most likely solve this issue and increase the performance.

## 3.2 Features Extraction from Pre-Trained CNN Approach

The second type of approach attempted was to take a CNN network (MobileNetV2) pre-trained on ImageNet, cut the output layer, and use the last layer as a feature extractor for the dataset's images. These feature arrays could then be used to train a SVM classifier.
Given the great results of the previous method, it was hard to say whether or not this approach could improve it. It was still decided to attempt it because, given the small number of images in the dataset, this feature-based approach could help train a model that is able to generalize much better.

A function was created to extract the features from the train set images while applying a number of random non-repeating rotations (up to 4) to each image. This is because in the previous method, during each epoch the images are seen again by the model, but possibly with different rotations since those are randomly chosen while the images are imported batch per batch. To give this method the best chance to succeed, this parameter was set to 4 to get the features of every possible rotation from each image.

A grid search with cross-validation was applied to the SVM model to find the best hyperparameters. The best model is then retrained using the combination of the train and validation data (since validation images are not needed to analyze overfitting, and to give this method the best chance).

### 3.2.1 Results and Evaluation

Table 3: Classification Reports for SVM Models

| **Class** | **SVM Best Parameters** | | | **SVM with Added Val Data** | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1-Score** | **Precision** | **Recall** | **F1-Score** |
| 0 | 0.98 | 0.98 | 0.98 | 0.98 | 0.99 | 0.99 |
| 1 | 0.64 | 0.60 | 0.62 | 0.68 | 0.62 | 0.65 |
| 2 | 0.98 | 0.97 | 0.98 | 0.99 | 0.97 | 0.98 |
| 3 | 0.63 | 0.68 | 0.65 | 0.65 | 0.72 | 0.68 |
| **Accuracy** | 0.81 | | | 0.82 | | |
| **Macro Avg** | 0.81 | 0.81 | 0.81 | 0.82 | 0.82 | 0.82 |
| **Weighted Avg** | 0.81 | 0.81 | 0.81 | 0.82 | 0.82 | 0.82 |

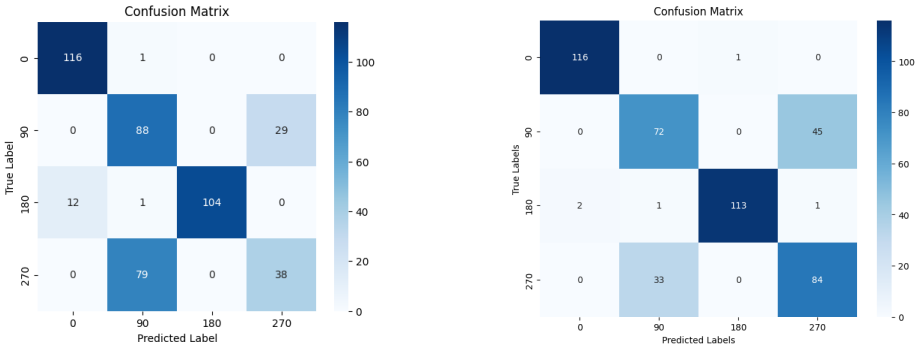


Figure 6: SVM Confusion Matrices

Parameters part of the grid search (best ones in bold): **C**: 0.1, 1, **10**, 100; **gamma**: **'scale'**, 'auto', 0.001, 0.01, 0.1, 1; **kernel**: 'linear', **'rbf'**.


Figure 7: SVM Errors on 0° and 180° Classes

### 3.2.2 Discussion

The performance is sharply lower than the hand-made CNN's. What is most interesting though, is the confusion matrix. It is very clear how the model is struggling to distinguish between the 90° and the 270° images. This is most likely due to a lack of information within the feature arrays, that are enough to describe the difference between straight and upside-down images (probably due to factors like the sky being in the above section of the images), but not enough to tell apart 90° and 270° rotations.
The addition of the validation set was worth a very minor performance increase. Although the validation set does not include many images, this still indicates that the size itself of the dataset isn't the problem for this method. The issue is solely the lack of information in the feature arrays to distinguish the two side-rotated image classes.
It is also intriguing to analyze which errors the model made on the 0° and 180° classes [Figure 7], since the model only made 5 misclassifications. Interestingly, the two repeated images are different from the previous method. However, this cannot be taken as an indication of better generalization, and it is also not a fair comparison since the hand-made CNN approach did not use the validation set as part of the training data.

## 3.3 Fine-Tuned CNN Approach

The last approach attempted was the fine-tuning of a CNN model pre-trained on ImageNet (the same MobileNetV2 used in the previous method) by freezing the network's weights, cutting the output layer, and replacing it with a

dense layer, followed by the custom output layer with 4 neurons (4 one-hot encoded classes). This method helps solve the problem of having a small dataset and provides a "hot start" and better generalization. For this reason, before starting this project, it was hypothesized to be the best method.

In this case, input images are pre-processed using the MobileNetV2 pipeline instead of simple normalization (contrary to the hand-made model).

Table 4: Fine-Tuned Model's Architecture Overview

| **Layer** | **Output Shape** | **Param #** |
|---|---|---|
| input | (224, 224, 3) | / |
| MobileNetV2 | (7, 7, 1024) | 7,037,504 |
| global_average_pooling2d | (1024) | 0 |
| dense_1 + Dropout (60%) | (512) | 524,800 |
| dense_2 | (4) | 2,052 |

Several models were tested, with initial results being suboptimal. Subsequent models improved performance by unfreezing weights, reducing the learning rate, and progressively fine-tuning further.

### 3.3.1 Results and Evaluation

Table 5: Classification Reports for Fine-Tuned Models

| **Class** | **Model 1 (frozen)** | | | **Model 2 (unfrozen)** | | |
|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **F1-Score** | **Precision** | **Recall** | **F1-Score** |
| 0 | 0.91 | 0.99 | 0.95 | 0.89 | 0.99 | 0.94 |
| 90 | 0.52 | 0.75 | 0.62 | 0.62 | 1.00 | 0.77 |
| 180 | 1.00 | 0.89 | 0.94 | 1.00 | 0.86 | 0.93 |
| 270 | 0.57 | 0.32 | 0.41 | 1.00 | 0.41 | 0.58 |
| **Accuracy** | 0.74 | | | 0.82 | | |
| **Class** | **Model 3 (unfrozen)** | | | **Model 4 (unfrozen)** | | |
| | **Precision** | **Recall** | **F1-Score** | **Precision** | **Recall** | **F1-Score** |
| 0 | 0.94 | 0.99 | 0.96 | 0.97 | 0.97 | 0.97 |
| 90 | 0.85 | 0.99 | 0.91 | 0.94 | 0.94 | 0.94 |
| 180 | 1.00 | 0.94 | 0.97 | 0.95 | 0.99 | 0.97 |
| 270 | 0.99 | 0.82 | 0.90 | 0.96 | 0.92 | 0.94 |
| **Accuracy** | 0.94 | | | 0.96 | | |

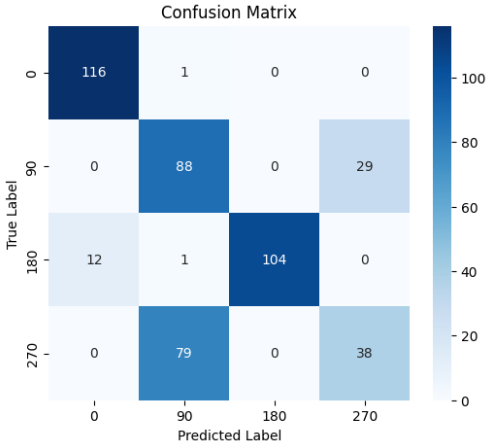


Figure 8: FT Model 1 Conf. Matrix

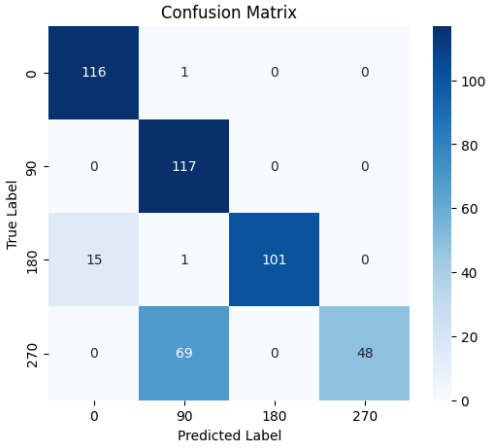


Figure 9: FT Model 2 Conf. Matrix

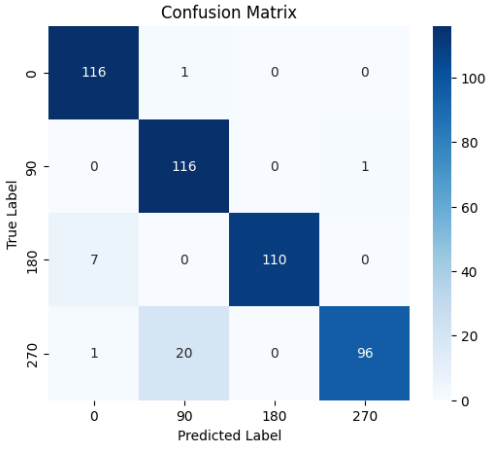


Figure 10: FT Model 3 Conf. Matrix

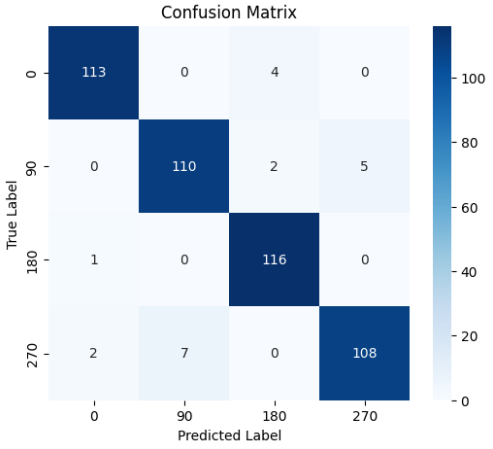


Figure 11: FT Model 4 Conf. Matrix


Figure 12: Best Fine-Tuned Model Errors on 0° and 180° Classes

### 3.3.2 Discussion

Training the model further slightly improves performance [Table 5], though it still doesn't match the hand-made CNN's [Table 2]. The first model struggles the same way the feature-based approach did. The more the unfrozen model is fine-tuned, the fewer mistakes are made between the 90° and the 270° classes.
Interestingly, the last model correctly classified image 58, which the hand-made CNN struggled with, but this doesn't necessarily suggest a better generalization due to the initialization, since other images are mislabeled across different rotations.
With better tuning of the added layers and optimized overfitting, this model could potentially match or outperform the hand-made CNN. However, given the computational effort and time required, it was decided to focus on other ideas for this project instead.

# 4 Predicting Full-Range Rotations

The second half of this project extends the best of the previously explored methods to the task of predicting the full-range rotations of images, meaning that images can have any rotation from 0° to 360°.

## 4.1 Hand-Crafted CNN Approach

The hand-crafted CNN model, trained from scratch, was chosen as it demonstrated the best performance in the previous section.

The dataset preparation for this problem is more complicated, since just rotating images by a random amount would create images with black patches that fill the empty areas created by non-quarter rotations, and this added information would be leveraged by the model to learn the rotations, which would cause it to not work on a real test set.
The idea was to crop the biggest square that fits inside of the rotated image, cutting off extra parts of the image but retrieving a rotated image without distortions or artifacts. To calculate the size of the square, this formula [3] was used.

Figure 13: Original

Figure 14: Rotated 30°

Figure 15: Cropped

Another challenge was the model's output, as the task shifted to a regression problem to determine the exact rotation. The issue was that using the degree values as labels treated rotations like 1° and 259° as opposites, even though they are nearly equivalent in a circular space. This discontinuity would most likely cause the model to struggle predicting rotations near this range.
To remove this discontinuity, the rotation values were converted to two $x$ and $y$ values (1), transforming the model's output shape accordingly.

$$x = \cos(\theta) \qquad y = \sin(\theta) \tag{1}$$

where $\theta$ is the rotation angle in radians.

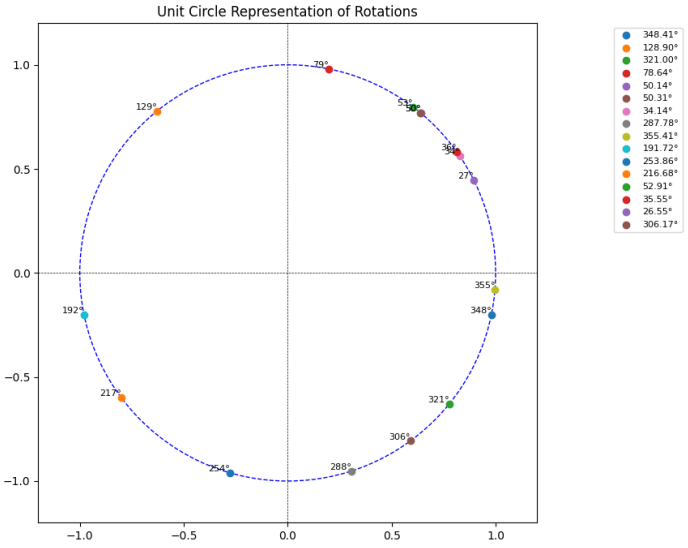


Figure 16: Rotation conversion examples


Figure 17: Rotation example images

This solution not only fixes the discontinuity, but also enhances the differences between opposite rotations.
The model's architecture was kept the same [Table 1], only the learning rate was slightly increased from *0.0001* to *0.0002* in the best model. Mean squared error (MSE) was chosen as loss function since it gives more weight to larger errors due to the squaring.
Considering the small size of the dataset and that not necessarily every image is perfectly level, it was not expected for this model to reach perfect results.

## 4.2 Results and Evaluation


Figure 18: Full-range rotation predictions


Figure 19: Full-range rotation correction

Loss (MSE): **0.1524**; Average Difference in Degrees: **27.08°**

### 4.3 Discussion

While the results are not perfect, an average error of 27° is fairly acceptable, especially considering the size of the dataset and the assumption that all images are perfectly level. With a larger, more carefully curated dataset and further optimization of the model (whether through architectural changes or exploring alternative approaches) significantly better results are achievable. While the average error might seem small, applying corrections [Figure 19] reveals that even slight discrepancies are quite noticeable to the human eye.

## 5 Conclusions

This project explored CNN-based methods for predicting image rotations, starting with quarter-turn rotations and extending to full-range rotations.
The hand-crafted CNN model, trained from scratch, outperformed feature extraction and fine-tuning approaches, achieving an impressive 98% accuracy on quarter-turn rotations. The model was adapted for full-range rotation prediction, where it achieved an average error of 27°, showcasing its versatility and effectiveness across different rotation tasks.
The success of the hand-crafted model was unexpected, but it highlighted the limitations of pre-trained models in capturing rotation-specific features. While the results are promising, future work should focus on expanding the dataset and exploring advanced techniques to further improve performance and generalization, particularly for the more challenging task of full-range rotation prediction.

## References


[1] A. Rougetet, "Landscape Pictures," Kaggle Dataset, 2020, https://www.kaggle.com/datasets/arnaud58/landscape-pictures.

[2] Heonh0, "Unpaired Day and Night cityview images," Kaggle Dataset, 2022, https://www.kaggle.com/datasets/heonh0/daynight-cityview.

[3] heropup, "Calculate dimensions of square inside a rotated square," Math Stack Exchange Forum, 2014, https://math.stackexchange.com/questions/828878/calculate-dimensions-of-square-inside-a-rotated-square.