Country classification

Artificial intelligence project
236502

Advisor: Alona Levy

Submitters:
Naser Albasar נסר אלבסאר
322244930
naser.al@campus.technion.ac.il

Nabeeh Abu Saleh נביה אבו סאלח
211518147
nabeeh@campus.technion.ac.il

<u>Table of Contents:</u>

Background:

Google street view covers a large part of earth, mostly covering the Americas, Europe and different chunks of Asia and Oceana, letting us roam the streets of many different countries.
Each country has many distinct features, some obvious such as unique landmarks (Big Ben, Eiffel Tower) and others subtle like the road signage and flora (Kiwi warning sign of New Zealand, Pegasus crossing sign of the UK and Peru), this fact gave rise to a web browser game called GeoGuessr (not a typo), released on May 2013, the games "classic" mode has 5 rounds, each one puts the player on a random street in Google street view and has the player guess where on earth are they, the more accurate the guess the more points are rewarded, another mode in the game follows the same concept but has the player guess what country they are in with no regard to distance.
In our project we will try to develop an algorithm/model that given a picture from Google street view as an input will predict which country the image was taken from.

Methodology:

We will attempt to solve the problem using learning algorithms and neural networks, since the problem in question involves image processing and classifying them, neural networks seem to fit this problem.
Some popular players of the game are able to consistently pinpoint the location of an images within seconds by noticing different small features in the picture, we hope to mimic this ability by training a neural network.
To train a model we would need data.

Planning:

Not all counties are covered equally on Google Street, and some are too small to randomize different locations from, hence why we needed to choose which countries we wanted to include in the project, we decided on 30 countries that we felt gave a mix of sufficient coverage, variety and scale.

Data collection:

We would need a way of collecting coordinated from different countries and another way of saving the image of the location specified by the coordinate.
For collecting coordinates we used the '3geonames' API which receives an Alpha-2 code (2 letter abbreviation for a countries name, example: US for United states, MX for Mexico) and outputs a JSON file containing the geological information of a random location in the country, we are interested in the 'latt' and 'longt' fields (latitude and longitude).
After we have the coordinates we need to get the images associated with the coordinates, for this we use Googles 'Maps' API, but before requesting the image from the exact coordinates we need to check if said coordinate is covered by Google, luckily the 'Maps' API has this option, if the coordinate isn't covered we request the nearest covered coordinate, after making sure the new coordinate are in the same country using '3geonames' reverse search function we can now request the image from 'Maps', save it with a name of (<Alpha-2 code>_<number>) this acts as the label of the image, then after move onto the next coordinate.
*notes: 'Maps' gives us the option of field of view, heading and pitch (looking up or down), we used fixed values for FOV (90) and pitch (0) but randomized the heading value (0-359), we also made sure that all of the images where taken by Google and not independent contributors which aren't granted to be on a

road or in a public area.
Of course API's aren't always available when we want them and aren't meant to be used in the frequency we are using them, especially '3geonames' which is hosted on a free website and will deny usage if we spam it repeatedly, furthermore, 'Maps' is a paid API, luckily Google offers a free trial, we would like to stay within this free trial as the cost could easily accumulate when running scripts overnight, thus we wrote a python script that makes sure that we aren't spamming '3geonames' by waiting 2 second whenever a request is denied before continuing to collect our coordinates, and another script the collects the images while reporting how many API calls we made as to make sure we aren't spinning our wheels going nowhere while making repeated API calls.
Once we had 1000 images from each 30 countries we duplicate the images and flipped them horizontally to artificially boost our data size, this could lead to training leakage, we will touch on this again in the models training section, in each experiment we maintained an equal ratio of countries in each of the splits.

Models:

CNN:

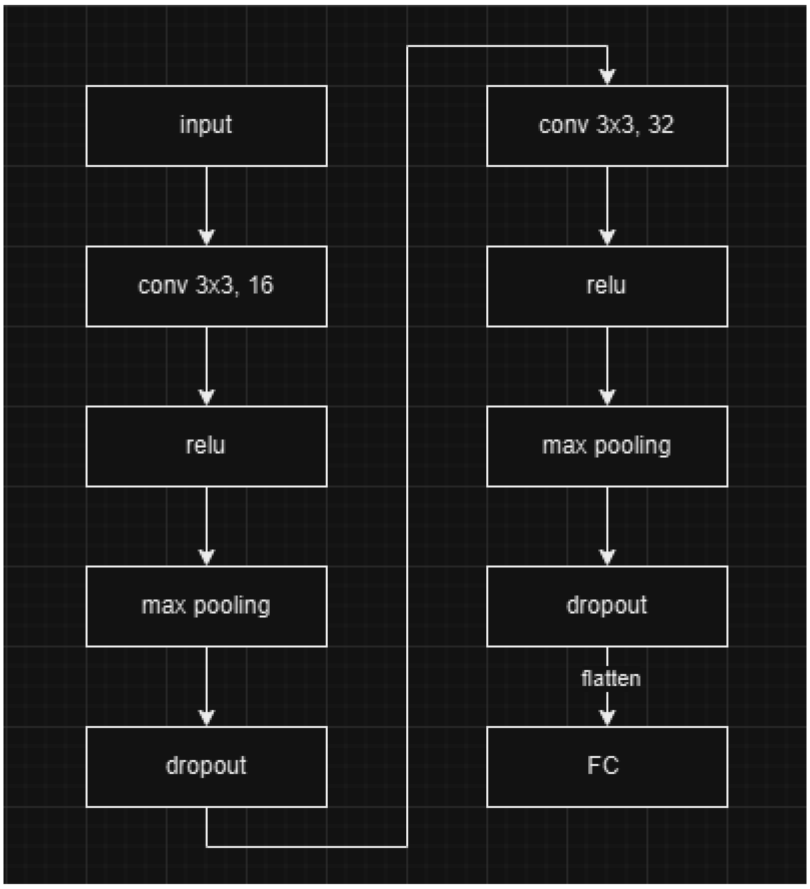
Before we use complex models we would like to have a benchmark of the simpler models, and so we used the most basic model that had a chance of yielding good results, that is a CNN model, which utilizes different number of convolution layers (we used 3x3 kernel with 1 stride and 1 padding) which are responsible for feature extraction and dimensionality reduction, a RELU activation, max pooling layers (2x2 size with a stride of 2), which is responsible for further dimensionality reduction and helps reduce overfitting, dropout layers (0.5 dropout rate) which further helps prevent overfitting, and finally a fully connected layer.
the flow is as follows:

$$conv \rightarrow relu \rightarrow \max pool \rightarrow dropout$$

repeated as many times as specified (we tried 2 and 3 layers, more detail in the experiments section), and then finishing with:

$$flatten \rightarrow fully\ connected$$

example for the model we used with 2 conv layers:

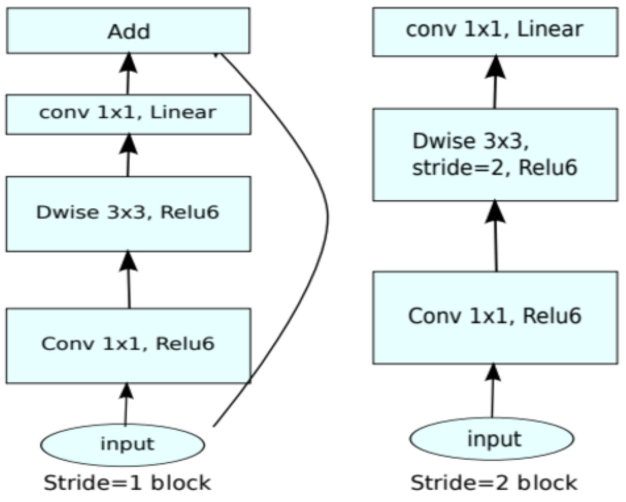


We used another model with 3 conv layers which add another conv layer starting with conv 3x3, 64 and continues like the others.

MobileNetV2:

[1] [2] MobileNetV2 is a 53-layer deep CNN model built for efficiency and speed, making it perfect for tasks like augmented reality, face recognition, and even medical image analysis. One of its best features is how lightweight it is, thanks to its design. Instead of using standard convolutions, MobileNetV2 uses depth-wise separable convolutions (a method that processes each input channel individually before combining them). This reduces the computational complexity without sacrificing accuracy.

The architecture also uses an inverted residual structure, where so called bottleneck layers (thin input and output layers) surround an expanded (thicker) intermediate layer. This is the opposite of traditional models like ResNet (which is mentioned below), which typically expand features in the input. Combined with stride-based blocks (one preserving input size, the other decreasing it's size), MobileNetV2 balances efficiency and feature extraction.



(d) Mobilenet V2 **[3]**

For this project, MobileNetV2's lightweight design has a good advantage. Its lower computational cost saved time during training and testing. By focusing on efficiency over depth, it seems like a good fit for our task, as it could reduce training times and maintain good accuracy.

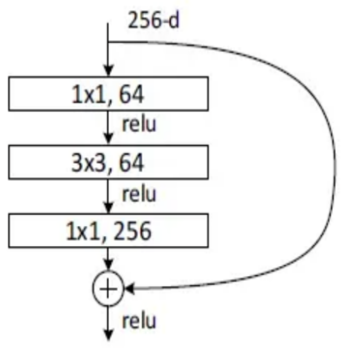
ResNet-50:

[4] ResNet-50 (Residual Network) is a CNN based architecture, which is designed to deal with deep neural networks problems and is known for its depth and efficiency in image classification tasks similar to our project.

The main problem ResNet solved was the degradation problem in deep neural networks. When networks become deeper, their accuracy decreases. This decrease is caused by the difficulty of optimizing the training process.

ResNet solved this problem using Residual Blocks (a series of convolutional layers followed by a shortcut connection (skip connection) that adds the block's input directly to its output), helping combat the degradation problem.

The residual block used in ResNet-50 is called the "Bottleneck Residual Block" which has the following architecture:



The Bottleneck Residual Block for ResNet-50/101/152.

[4]

ResNet comes in many depths, such as ResNet-(18, 32, 101, 152, etc.), but we have decided to use ResNet-50.

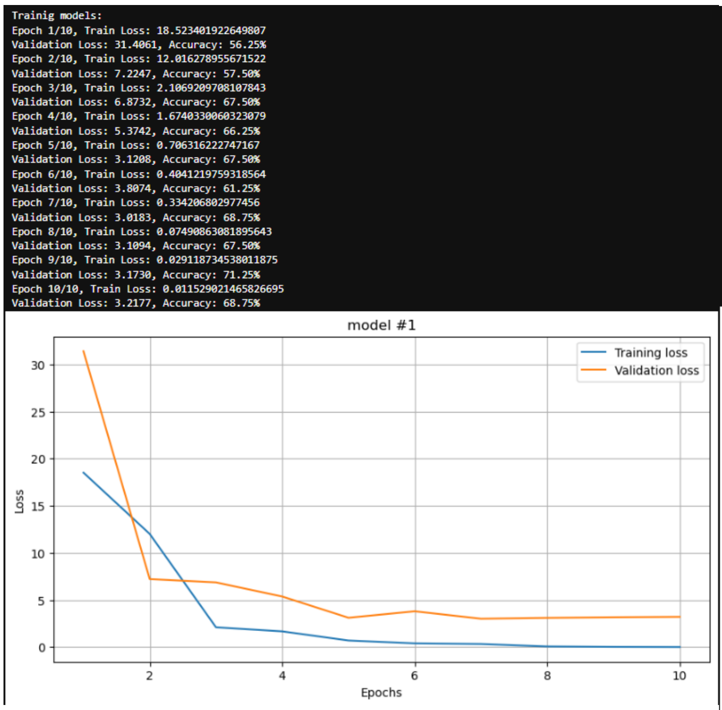
This architecture seems promising as its ability to capture complex features with limited data sizes and its anti-overfitting qualities will prove useful.

Experiments:

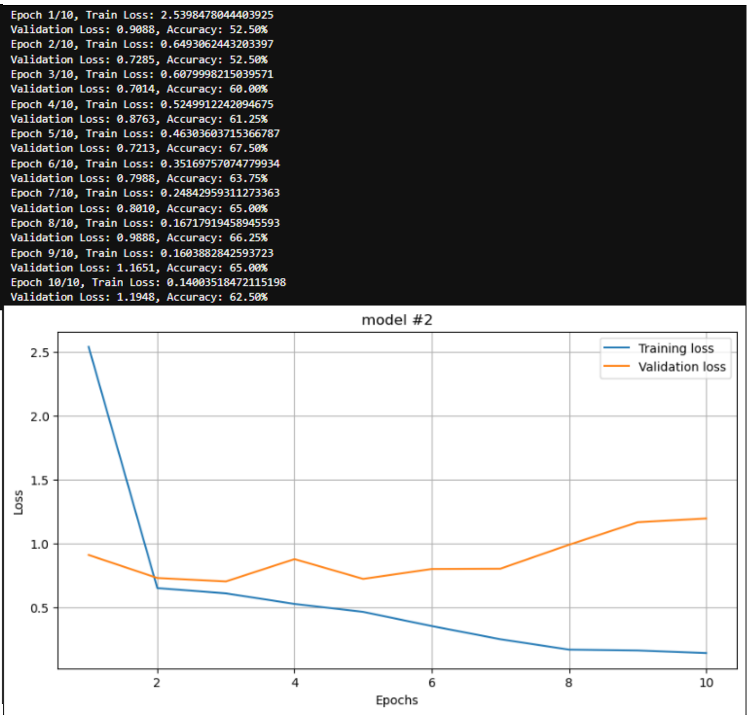
Proof of concept:

Before confirming the project idea we wanted to gauge how suited is our solution, and so for our first experiment we took 200 images from Finland and 200 images from Mexico, the question we wanted to answer was would a simple model like CNN with minimal convolution layers be able to pick up on the obvious differences between two opposite climates? The differences being the cold and wet climate of Finland vs the hot and dry climate of Mexico.
The first model we used for this experiment was a one convolution layer CNN and the second model was a two convolution layers CNN, we used the Adam optimizer and cross entropy loss function for 10 training epochs and a-80% 20% train validation split.

Results:



```
Trainig models:
Epoch 1/10, Train Loss: 18.523401922649807
Validation Loss: 31.4061, Accuracy: 56.25%
Epoch 2/10, Train Loss: 12.016278955671522
Validation Loss: 7.2247, Accuracy: 57.50%
Epoch 3/10, Train Loss: 2.1069209708107843
Validation Loss: 6.8732, Accuracy: 67.50%
Epoch 4/10, Train Loss: 1.6740330060323079
Validation Loss: 5.3742, Accuracy: 66.25%
Epoch 5/10, Train Loss: 0.706316222747167
Validation Loss: 3.1208, Accuracy: 67.50%
Epoch 6/10, Train Loss: 0.4041219759318564
Validation Loss: 3.8074, Accuracy: 61.25%
Epoch 7/10, Train Loss: 0.334206802977456
Validation Loss: 3.0183, Accuracy: 68.75%
Epoch 8/10, Train Loss: 0.07490863081895643
Validation Loss: 3.1094, Accuracy: 67.50%
Epoch 9/10, Train Loss: 0.029118734538011875
Validation Loss: 3.1730, Accuracy: 71.25%
Epoch 10/10, Train Loss: 0.011529021465826695
Validation Loss: 3.2177, Accuracy: 68.75%
```



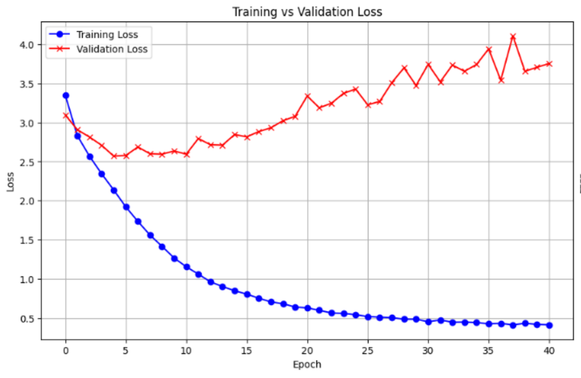
```
Epoch 1/10, Train Loss: 2.5398478044403925
Validation Loss: 0.9088, Accuracy: 52.50%
Epoch 2/10, Train Loss: 0.6493062443203397
Validation Loss: 0.7285, Accuracy: 52.50%
Epoch 3/10, Train Loss: 0.6079998215039571
Validation Loss: 0.7014, Accuracy: 60.00%
Epoch 4/10, Train Loss: 0.5249912242094675
Validation Loss: 0.8763, Accuracy: 61.25%
Epoch 5/10, Train Loss: 0.46303603715366787
Validation Loss: 0.7213, Accuracy: 67.50%
Epoch 6/10, Train Loss: 0.35169757074779934
Validation Loss: 0.7988, Accuracy: 63.75%
Epoch 7/10, Train Loss: 0.24842959311273363
Validation Loss: 0.8010, Accuracy: 65.00%
Epoch 8/10, Train Loss: 0.16717919458945593
Validation Loss: 0.9888, Accuracy: 66.25%
Epoch 9/10, Train Loss: 0.1603882842593723
Validation Loss: 1.1651, Accuracy: 65.00%
Epoch 10/10, Train Loss: 0.14003518472115198
Validation Loss: 1.1948, Accuracy: 62.50%
```

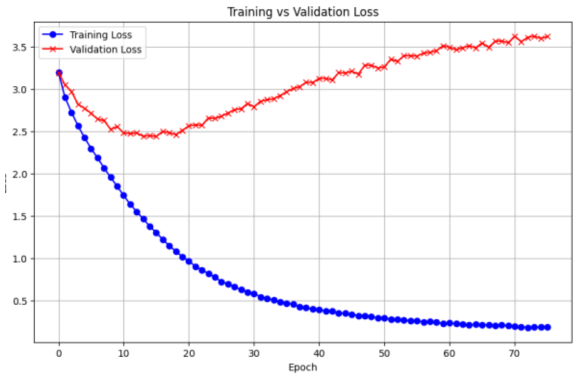
from these results we can conclude that this approach would work, as we can see the simplest model reached an accuracy of 71.25% with only 160 images in the training set for each class and 10 training epochs and without hyper-tunning of the learning rate or any other hyperparameters, thus we can say that our solution has the capacity to solve the problem.

CNN Benchmark:

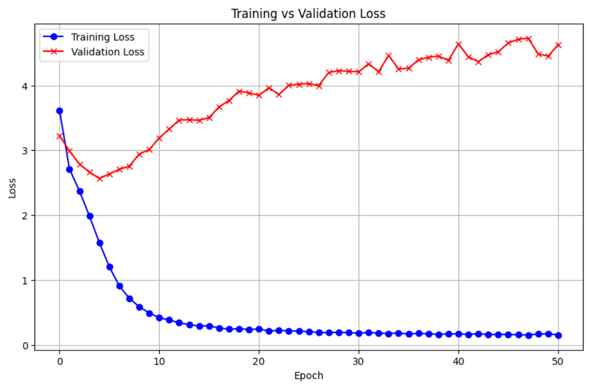
Now we would like to see how our basic model performs on a larger dataset, for this experiment we used a 2 layer and 3 layer CNN model, each layer has an appropriate size (3x3 kernel, 1stride, 1 padding), RELU activation, max pooling and dropout, just like we described in the methodology section.
here we had 1000 images from each country, using a 80-20 split and tuning the learning rate, the question we would like to answer was; would a simple CNN model be capable of solving the problem?

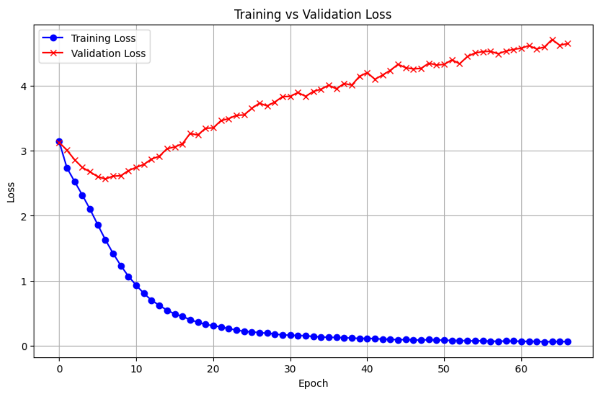
Results:
3 layers:



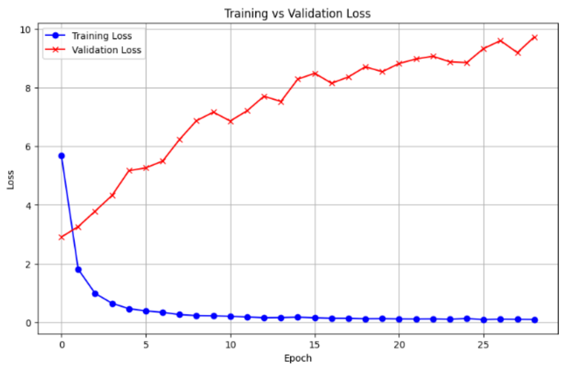


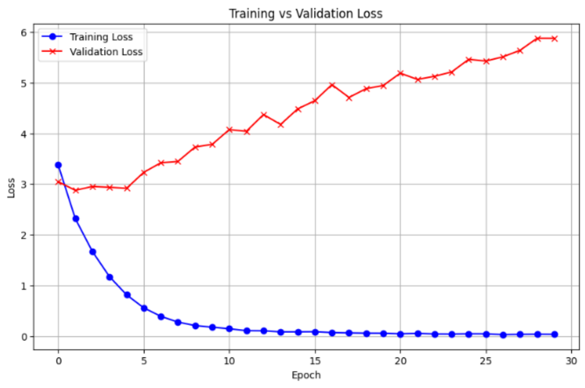
The graph on the left is the result of running a 3 layered CNN with 0.001 learning rate and the one on the right ran with 0.0001 learning rate, in both we see a clear case of overfitting, caricaturized by a large gap between the training and validation loss, furthermore, the training accuracy reached was 96% while the best validation accuracy reached was 36.67%, further indicating overfitting, this might be cause by the complexity of our model, enabling our model to "learn" out training data by remembering it rather than finding any generalizations, let's see how a less complicated model performs.
2 layers:





again, the graph on the left is the result of running a 2 layered CNN with 0.001 learning rate and the one on the right ran with 0.0001 learning rate, in both we see that we are overfitting to our data, as the training accuracy reach 95% and 97% the highest the validation accuracy was 30.9%, either our model is still too complex, our architecture isn't best suited for the task or we don't have enough data.
Out of curiosity we tried running the same experiment with a 1-layer CNN, these were the results:





Again, we see overfitting, training accuracy reached 99% while the validation accuracy didn't cross 25%, also we see that the validation loss only grew after a couple of training epochs, further indicating overfitting, we can conclude that either this architecture isn't best suited for the task, or we don't have enough data.
Moving forward, we would like to achieve a validation accuracy of more than 36.66% with other architectures.

MobileNetV2 Benchmark:

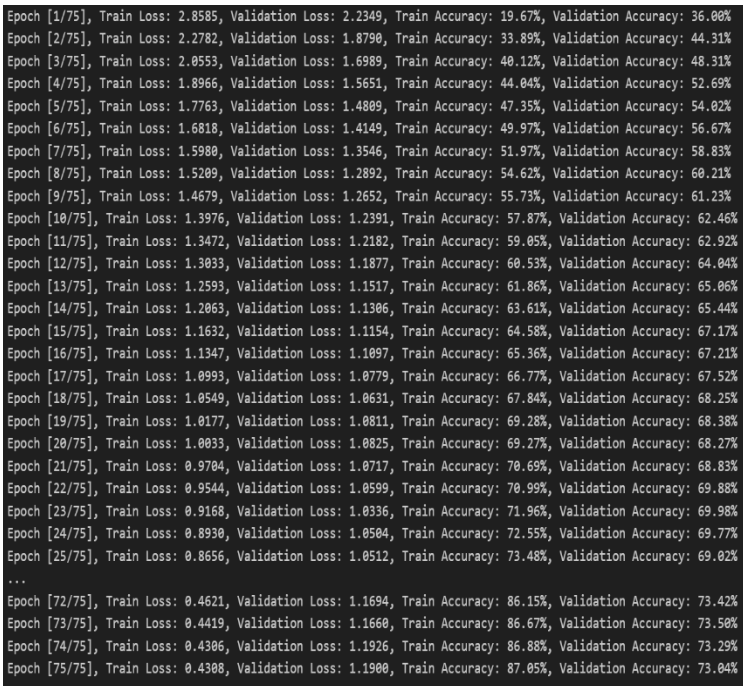
With MobileNet, we have conducted multiple experiments, to find the best configuration for improving over the CNN models.

For these experiments we used an- 80 20 train test split, and within the train set we split it further (80 20) into train validation splits to monitor overfitting, thus giving us a 64% 16% 20% train validation test split, this is the split we will be using for the rest of the experiments.

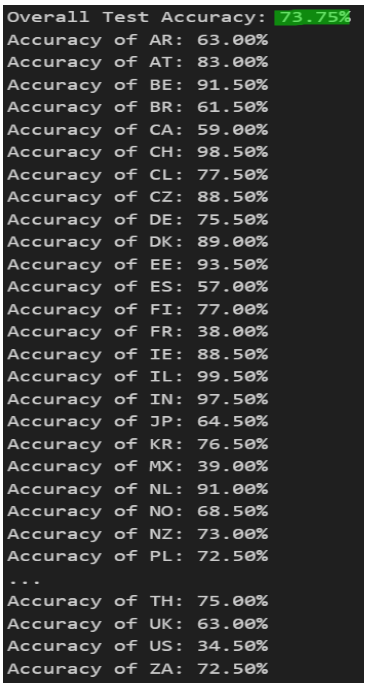
For data transformation (transform), we have used different transformations for the training set, and testing set. For the testing set, we have resized the pictures into 224x224 (which is the size appropriate for MobileNet) and normalized the data accordingly.

On the other hand, for training set, we used a similar transformation to the test set, in addition to random horizontal flips and random rotations to introduce data augmentation, which increased dataset diversity. Furthermore, we used a cropping method where given a picture from the dataset, it crops a random 224x224 part of the picture for training instead of the whole one.

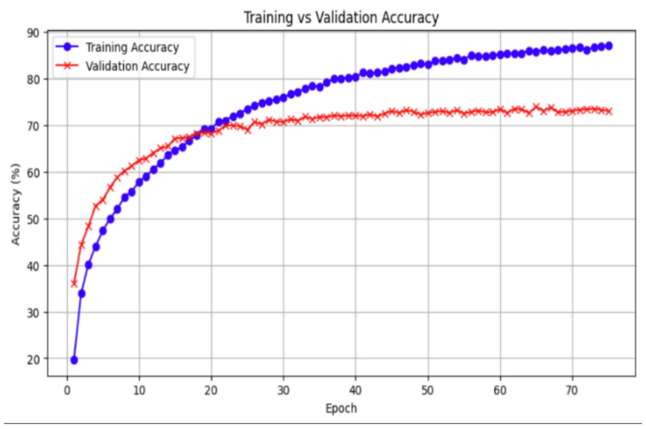
Results:

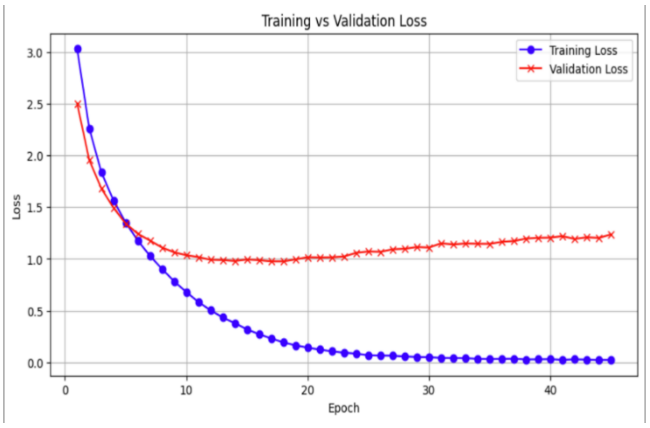


```
Epoch [1/75], Train Loss: 2.8585, Validation Loss: 2.2349, Train Accuracy: 19.67%, Validation Accuracy: 36.00%
Epoch [2/75], Train Loss: 2.2782, Validation Loss: 1.8790, Train Accuracy: 33.89%, Validation Accuracy: 44.31%
Epoch [3/75], Train Loss: 2.0553, Validation Loss: 1.6989, Train Accuracy: 40.12%, Validation Accuracy: 48.31%
Epoch [4/75], Train Loss: 1.8966, Validation Loss: 1.5651, Train Accuracy: 44.04%, Validation Accuracy: 52.69%
Epoch [5/75], Train Loss: 1.7763, Validation Loss: 1.4809, Train Accuracy: 47.35%, Validation Accuracy: 54.02%
Epoch [6/75], Train Loss: 1.6818, Validation Loss: 1.4149, Train Accuracy: 49.97%, Validation Accuracy: 56.67%
Epoch [7/75], Train Loss: 1.5980, Validation Loss: 1.3546, Train Accuracy: 51.97%, Validation Accuracy: 58.83%
Epoch [8/75], Train Loss: 1.5209, Validation Loss: 1.2892, Train Accuracy: 54.62%, Validation Accuracy: 60.21%
Epoch [9/75], Train Loss: 1.4679, Validation Loss: 1.2652, Train Accuracy: 55.73%, Validation Accuracy: 61.23%
Epoch [10/75], Train Loss: 1.3976, Validation Loss: 1.2391, Train Accuracy: 57.87%, Validation Accuracy: 62.46%
Epoch [11/75], Train Loss: 1.3472, Validation Loss: 1.2182, Train Accuracy: 59.05%, Validation Accuracy: 62.92%
Epoch [12/75], Train Loss: 1.3033, Validation Loss: 1.1877, Train Accuracy: 60.53%, Validation Accuracy: 64.04%
Epoch [13/75], Train Loss: 1.2593, Validation Loss: 1.1517, Train Accuracy: 61.86%, Validation Accuracy: 65.06%
Epoch [14/75], Train Loss: 1.2063, Validation Loss: 1.1306, Train Accuracy: 63.61%, Validation Accuracy: 65.44%
Epoch [15/75], Train Loss: 1.1632, Validation Loss: 1.1154, Train Accuracy: 64.58%, Validation Accuracy: 67.17%
Epoch [16/75], Train Loss: 1.1347, Validation Loss: 1.1097, Train Accuracy: 65.36%, Validation Accuracy: 67.21%
Epoch [17/75], Train Loss: 1.0993, Validation Loss: 1.0779, Train Accuracy: 66.77%, Validation Accuracy: 67.52%
Epoch [18/75], Train Loss: 1.0549, Validation Loss: 1.0631, Train Accuracy: 67.84%, Validation Accuracy: 68.25%
Epoch [19/75], Train Loss: 1.0177, Validation Loss: 1.0811, Train Accuracy: 69.28%, Validation Accuracy: 68.38%
Epoch [20/75], Train Loss: 1.0033, Validation Loss: 1.0825, Train Accuracy: 69.27%, Validation Accuracy: 68.27%
Epoch [21/75], Train Loss: 0.9704, Validation Loss: 1.0717, Train Accuracy: 70.69%, Validation Accuracy: 68.83%
Epoch [22/75], Train Loss: 0.9544, Validation Loss: 1.0599, Train Accuracy: 70.99%, Validation Accuracy: 69.88%
Epoch [23/75], Train Loss: 0.9168, Validation Loss: 1.0336, Train Accuracy: 71.96%, Validation Accuracy: 69.98%
Epoch [24/75], Train Loss: 0.8930, Validation Loss: 1.0504, Train Accuracy: 72.55%, Validation Accuracy: 69.77%
Epoch [25/75], Train Loss: 0.8656, Validation Loss: 1.0512, Train Accuracy: 73.48%, Validation Accuracy: 69.02%
...
Epoch [72/75], Train Loss: 0.4621, Validation Loss: 1.1694, Train Accuracy: 86.15%, Validation Accuracy: 73.42%
Epoch [73/75], Train Loss: 0.4419, Validation Loss: 1.1660, Train Accuracy: 86.67%, Validation Accuracy: 73.50%
Epoch [74/75], Train Loss: 0.4306, Validation Loss: 1.1926, Train Accuracy: 86.88%, Validation Accuracy: 73.29%
Epoch [75/75], Train Loss: 0.4308, Validation Loss: 1.1900, Train Accuracy: 87.05%, Validation Accuracy: 73.04%
```



```
Overall Test Accuracy: 73.75%
Accuracy of AR: 63.00%
Accuracy of AT: 83.00%
Accuracy of BE: 91.50%
Accuracy of BR: 61.50%
Accuracy of CA: 59.00%
Accuracy of CH: 98.50%
Accuracy of CL: 77.50%
Accuracy of CZ: 88.50%
Accuracy of DE: 75.50%
Accuracy of DK: 89.00%
Accuracy of EE: 93.50%
Accuracy of ES: 57.00%
Accuracy of FI: 77.00%
Accuracy of FR: 38.00%
Accuracy of IE: 88.50%
Accuracy of IL: 99.50%
Accuracy of IN: 97.50%
Accuracy of JP: 64.50%
Accuracy of KR: 76.50%
Accuracy of MX: 39.00%
Accuracy of NL: 91.00%
Accuracy of NO: 68.50%
Accuracy of NZ: 73.00%
Accuracy of PL: 72.50%
...
Accuracy of TH: 75.00%
Accuracy of UK: 63.00%
Accuracy of US: 34.50%
Accuracy of ZA: 72.50%
```

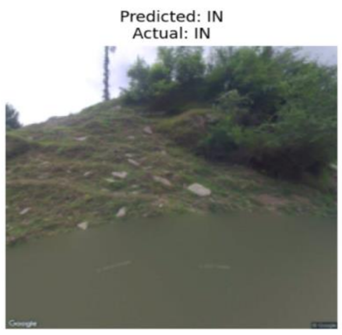




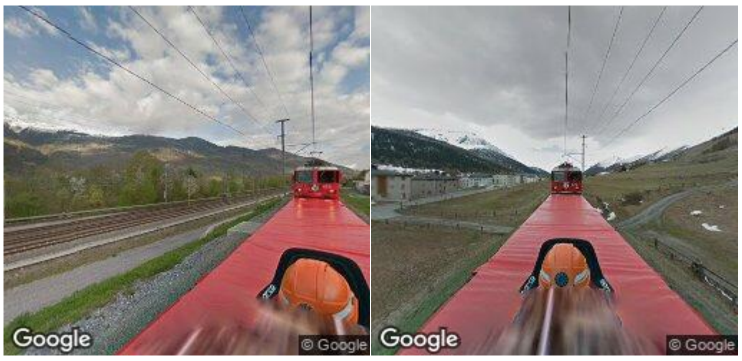
Compared to the previous architecture we have tried, MobileNetV2 resulted in much better results as expected, it reached 73.75% accuracy for testing which is a decent improvement.

This experiment's results were impressive in comparison to the models that we have mentioned before, since for test accuracy we have got ~73.75%, in contrast to the first experiments, we see that the validation loss didn't climb back up, from this we can conclude that this architecture is better suited for the task since there are no obvious indications of overfitting.

While testing the trained model we noticed that some countries saw a better accuracy, looking at our data we noticed that these countries had “markers” that the models might have picked up on, for instance we noticed that most of the images from India had the bottom third of the picture blurred, this blur is unique to India, it is most likely why the model achieved this high of an accuracy.



Another example was from Chile, some picture we taken from atop a moving train along its travel route, with the same prominent helmet and railroad car in the shot the model might have picked this up to, with some 30 unique images from the same train at different locations, this might have helped the model.

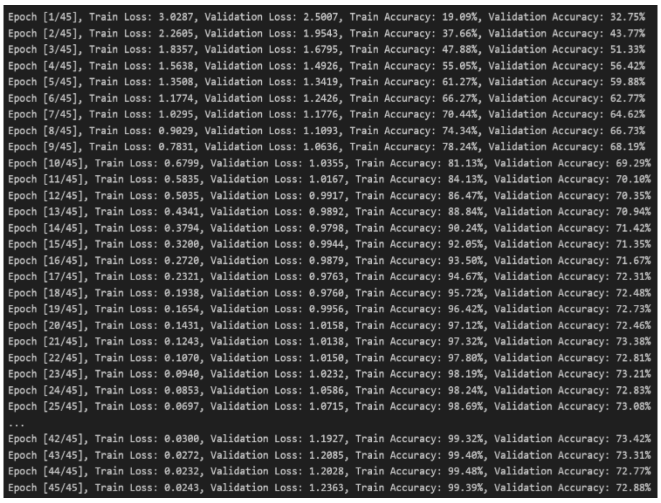


ResNet50 benchmark:

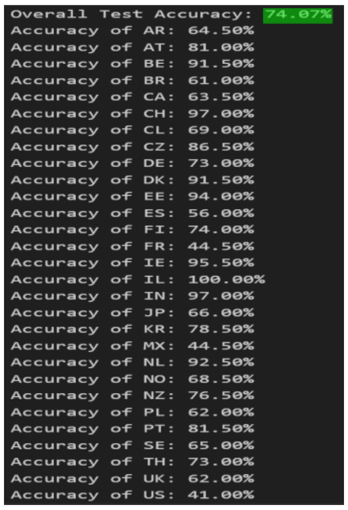
With ResNet50, we have used the same test transformation as MobileNetV2. We have tried multiple methods, after realizing that it could be the best architecture among the ones we tried, because of its strength dealing with a small dataset like ours, therefore, we conducted 4 experiments using it, and the results we have got were not disappointing.

Initially, the first experiment that we have tried is simply resizing the whole pictures in the training transformation to 224x224, it also included random horizontal flips and random rotations similar to MobileNet first experiment.

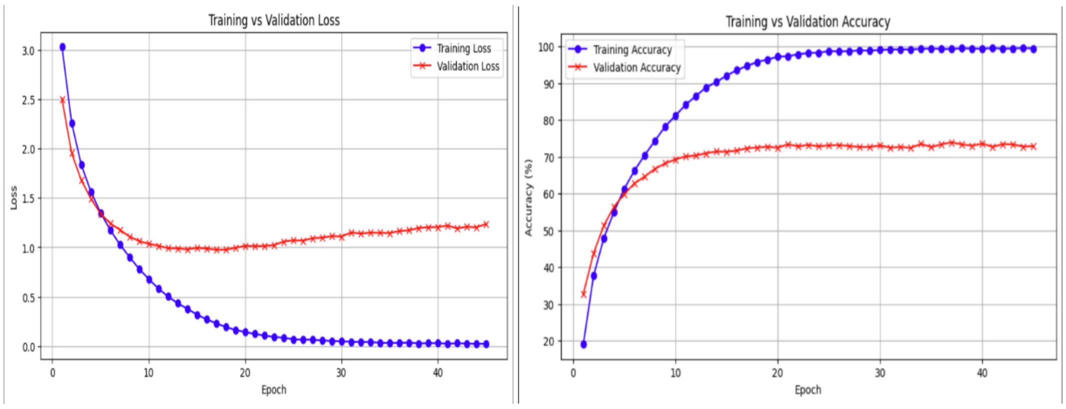
Results:



```
Epoch [1/45], Train Loss: 3.0287, Validation Loss: 2.5007, Train Accuracy: 19.09%, Validation Accuracy: 32.75%
Epoch [2/45], Train Loss: 2.2605, Validation Loss: 1.9543, Train Accuracy: 37.66%, Validation Accuracy: 43.77%
Epoch [3/45], Train Loss: 1.8357, Validation Loss: 1.6795, Train Accuracy: 47.88%, Validation Accuracy: 51.33%
Epoch [4/45], Train Loss: 1.5638, Validation Loss: 1.4926, Train Accuracy: 55.05%, Validation Accuracy: 56.42%
Epoch [5/45], Train Loss: 1.3508, Validation Loss: 1.3419, Train Accuracy: 61.27%, Validation Accuracy: 59.88%
Epoch [6/45], Train Loss: 1.1774, Validation Loss: 1.2426, Train Accuracy: 66.27%, Validation Accuracy: 62.77%
Epoch [7/45], Train Loss: 1.0295, Validation Loss: 1.1776, Train Accuracy: 70.44%, Validation Accuracy: 64.62%
Epoch [8/45], Train Loss: 0.9029, Validation Loss: 1.1093, Train Accuracy: 74.34%, Validation Accuracy: 66.73%
Epoch [9/45], Train Loss: 0.7831, Validation Loss: 1.0636, Train Accuracy: 78.24%, Validation Accuracy: 68.19%
Epoch [10/45], Train Loss: 0.6799, Validation Loss: 1.0355, Train Accuracy: 81.13%, Validation Accuracy: 69.29%
Epoch [11/45], Train Loss: 0.5835, Validation Loss: 1.0167, Train Accuracy: 84.13%, Validation Accuracy: 70.10%
Epoch [12/45], Train Loss: 0.5035, Validation Loss: 0.9917, Train Accuracy: 86.47%, Validation Accuracy: 70.35%
Epoch [13/45], Train Loss: 0.4341, Validation Loss: 0.9892, Train Accuracy: 88.84%, Validation Accuracy: 70.94%
Epoch [14/45], Train Loss: 0.3794, Validation Loss: 0.9798, Train Accuracy: 90.24%, Validation Accuracy: 71.42%
Epoch [15/45], Train Loss: 0.3200, Validation Loss: 0.9944, Train Accuracy: 92.05%, Validation Accuracy: 71.35%
Epoch [16/45], Train Loss: 0.2720, Validation Loss: 0.9879, Train Accuracy: 93.50%, Validation Accuracy: 71.67%
Epoch [17/45], Train Loss: 0.2321, Validation Loss: 0.9763, Train Accuracy: 94.67%, Validation Accuracy: 72.31%
Epoch [18/45], Train Loss: 0.1938, Validation Loss: 0.9760, Train Accuracy: 95.72%, Validation Accuracy: 72.48%
Epoch [19/45], Train Loss: 0.1654, Validation Loss: 0.9956, Train Accuracy: 96.42%, Validation Accuracy: 72.73%
Epoch [20/45], Train Loss: 0.1431, Validation Loss: 1.0158, Train Accuracy: 97.12%, Validation Accuracy: 72.46%
Epoch [21/45], Train Loss: 0.1243, Validation Loss: 1.0138, Train Accuracy: 97.32%, Validation Accuracy: 73.38%
Epoch [22/45], Train Loss: 0.1070, Validation Loss: 1.0150, Train Accuracy: 97.80%, Validation Accuracy: 72.81%
Epoch [23/45], Train Loss: 0.0940, Validation Loss: 1.0232, Train Accuracy: 98.19%, Validation Accuracy: 73.21%
Epoch [24/45], Train Loss: 0.0853, Validation Loss: 1.0586, Train Accuracy: 98.24%, Validation Accuracy: 72.83%
Epoch [25/45], Train Loss: 0.0697, Validation Loss: 1.0715, Train Accuracy: 98.69%, Validation Accuracy: 73.08%
...
Epoch [42/45], Train Loss: 0.0300, Validation Loss: 1.1927, Train Accuracy: 99.32%, Validation Accuracy: 73.42%
Epoch [43/45], Train Loss: 0.0272, Validation Loss: 1.2085, Train Accuracy: 99.40%, Validation Accuracy: 73.31%
Epoch [44/45], Train Loss: 0.0232, Validation Loss: 1.2028, Train Accuracy: 99.48%, Validation Accuracy: 72.77%
Epoch [45/45], Train Loss: 0.0243, Validation Loss: 1.2363, Train Accuracy: 99.39%, Validation Accuracy: 72.88%
```



```
Overall Test Accuracy: 74.07%
Accuracy of AR: 64.50%
Accuracy of AT: 81.00%
Accuracy of BE: 91.50%
Accuracy of BR: 61.00%
Accuracy of CA: 63.50%
Accuracy of CH: 97.00%
Accuracy of CL: 69.00%
Accuracy of CZ: 86.50%
Accuracy of DE: 73.00%
Accuracy of DK: 91.50%
Accuracy of EE: 94.00%
Accuracy of ES: 56.00%
Accuracy of FI: 74.00%
Accuracy of FR: 44.50%
Accuracy of IE: 95.50%
Accuracy of IL: 100.00%
Accuracy of IN: 97.00%
Accuracy of JP: 66.00%
Accuracy of KR: 78.50%
Accuracy of MX: 44.50%
Accuracy of NL: 92.50%
Accuracy of NO: 68.50%
Accuracy of NZ: 76.50%
Accuracy of PL: 62.00%
Accuracy of PT: 81.50%
Accuracy of SE: 65.00%
Accuracy of TH: 73.00%
Accuracy of UK: 62.00%
Accuracy of US: 41.00%
```



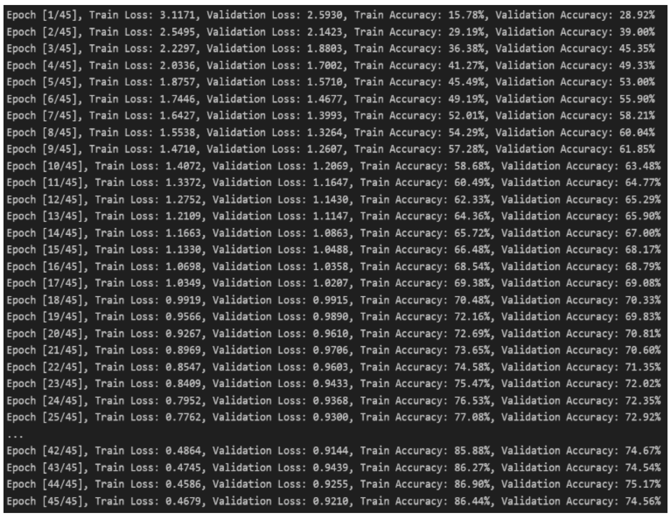
This experiment yielded good results, with slight improvement over MobileNetV2 experiments, by getting 74% test accuracy, would have guessed that this architecture would perform much better than MobileNetV2.

From here we wanted to improve our model, and so the purpose of the next experiments is to test some techniques that could improve the performance of the model.

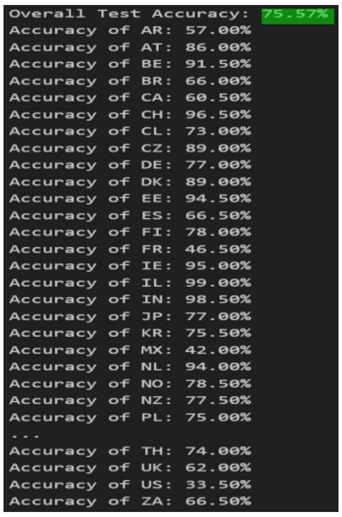
Cropping:

In this experiment we tried to change the training transformation, introducing cropping into the transformation, meaning that we take a random crop of the picture and resize it to the appropriate size (224x224), this should lead to better results as now we force the model to focus on specific features of the images.

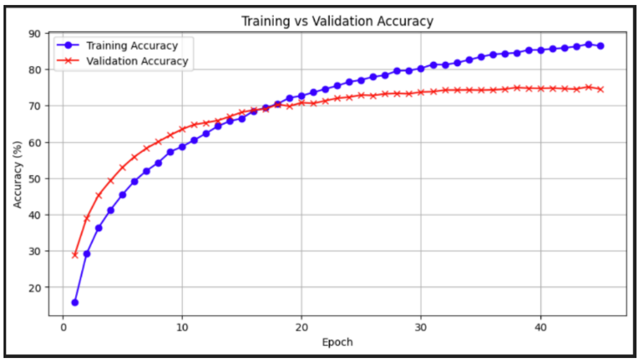
Results:

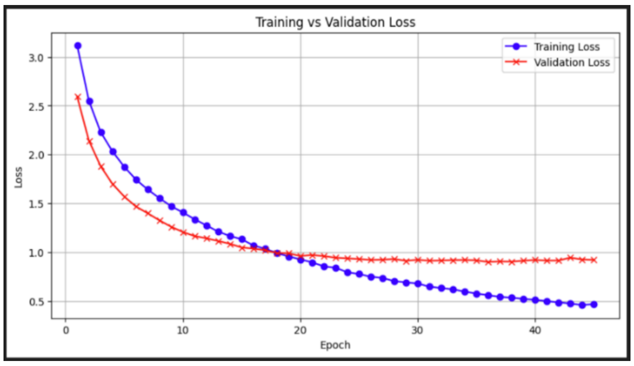


```
Epoch [1/45], Train Loss: 3.1171, Validation Loss: 2.5930, Train Accuracy: 15.78%, Validation Accuracy: 28.92%
Epoch [2/45], Train Loss: 2.5495, Validation Loss: 2.1423, Train Accuracy: 29.19%, Validation Accuracy: 39.00%
Epoch [3/45], Train Loss: 2.2297, Validation Loss: 1.8803, Train Accuracy: 36.38%, Validation Accuracy: 45.35%
Epoch [4/45], Train Loss: 2.0336, Validation Loss: 1.7002, Train Accuracy: 41.27%, Validation Accuracy: 49.33%
Epoch [5/45], Train Loss: 1.8757, Validation Loss: 1.5710, Train Accuracy: 45.49%, Validation Accuracy: 53.00%
Epoch [6/45], Train Loss: 1.7446, Validation Loss: 1.4677, Train Accuracy: 49.19%, Validation Accuracy: 55.90%
Epoch [7/45], Train Loss: 1.6427, Validation Loss: 1.3993, Train Accuracy: 52.01%, Validation Accuracy: 58.21%
Epoch [8/45], Train Loss: 1.5538, Validation Loss: 1.3264, Train Accuracy: 54.29%, Validation Accuracy: 60.04%
Epoch [9/45], Train Loss: 1.4710, Validation Loss: 1.2607, Train Accuracy: 57.28%, Validation Accuracy: 61.85%
Epoch [10/45], Train Loss: 1.4072, Validation Loss: 1.2069, Train Accuracy: 58.68%, Validation Accuracy: 63.48%
Epoch [11/45], Train Loss: 1.3372, Validation Loss: 1.1647, Train Accuracy: 60.49%, Validation Accuracy: 64.77%
Epoch [12/45], Train Loss: 1.2752, Validation Loss: 1.1430, Train Accuracy: 62.33%, Validation Accuracy: 65.29%
Epoch [13/45], Train Loss: 1.2109, Validation Loss: 1.1147, Train Accuracy: 64.36%, Validation Accuracy: 65.90%
Epoch [14/45], Train Loss: 1.1663, Validation Loss: 1.0863, Train Accuracy: 65.72%, Validation Accuracy: 67.00%
Epoch [15/45], Train Loss: 1.1330, Validation Loss: 1.0488, Train Accuracy: 66.48%, Validation Accuracy: 68.17%
Epoch [16/45], Train Loss: 1.0698, Validation Loss: 1.0358, Train Accuracy: 68.54%, Validation Accuracy: 68.79%
Epoch [17/45], Train Loss: 1.0349, Validation Loss: 1.0207, Train Accuracy: 69.38%, Validation Accuracy: 69.08%
Epoch [18/45], Train Loss: 0.9919, Validation Loss: 0.9915, Train Accuracy: 70.48%, Validation Accuracy: 70.33%
Epoch [19/45], Train Loss: 0.9566, Validation Loss: 0.9890, Train Accuracy: 72.16%, Validation Accuracy: 69.83%
Epoch [20/45], Train Loss: 0.9267, Validation Loss: 0.9610, Train Accuracy: 72.69%, Validation Accuracy: 70.81%
Epoch [21/45], Train Loss: 0.8969, Validation Loss: 0.9706, Train Accuracy: 73.65%, Validation Accuracy: 70.60%
Epoch [22/45], Train Loss: 0.8547, Validation Loss: 0.9603, Train Accuracy: 74.58%, Validation Accuracy: 71.35%
Epoch [23/45], Train Loss: 0.8409, Validation Loss: 0.9433, Train Accuracy: 75.47%, Validation Accuracy: 72.02%
Epoch [24/45], Train Loss: 0.7952, Validation Loss: 0.9368, Train Accuracy: 76.53%, Validation Accuracy: 72.35%
Epoch [25/45], Train Loss: 0.7762, Validation Loss: 0.9300, Train Accuracy: 77.08%, Validation Accuracy: 72.92%
...
Epoch [42/45], Train Loss: 0.4864, Validation Loss: 0.9144, Train Accuracy: 85.88%, Validation Accuracy: 74.67%
Epoch [43/45], Train Loss: 0.4745, Validation Loss: 0.9439, Train Accuracy: 86.27%, Validation Accuracy: 74.54%
Epoch [44/45], Train Loss: 0.4586, Validation Loss: 0.9255, Train Accuracy: 86.90%, Validation Accuracy: 75.17%
Epoch [45/45], Train Loss: 0.4679, Validation Loss: 0.9210, Train Accuracy: 86.44%, Validation Accuracy: 74.56%
```



```
Overall Test Accuracy: 75.57%
Accuracy of AR: 57.00%
Accuracy of AT: 86.00%
Accuracy of BE: 91.50%
Accuracy of BR: 66.00%
Accuracy of CA: 60.50%
Accuracy of CH: 96.50%
Accuracy of CL: 73.00%
Accuracy of CZ: 89.00%
Accuracy of DE: 77.00%
Accuracy of DK: 89.00%
Accuracy of EE: 94.50%
Accuracy of ES: 66.50%
Accuracy of FI: 78.00%
Accuracy of FR: 46.50%
Accuracy of IE: 95.00%
Accuracy of IL: 99.00%
Accuracy of IN: 98.50%
Accuracy of JP: 77.00%
Accuracy of KR: 75.50%
Accuracy of MX: 42.00%
Accuracy of NL: 94.00%
Accuracy of NO: 78.50%
Accuracy of NZ: 77.50%
Accuracy of PL: 75.00%
...
Accuracy of TH: 74.00%
Accuracy of UK: 62.00%
Accuracy of US: 33.50%
Accuracy of ZA: 66.50%
```





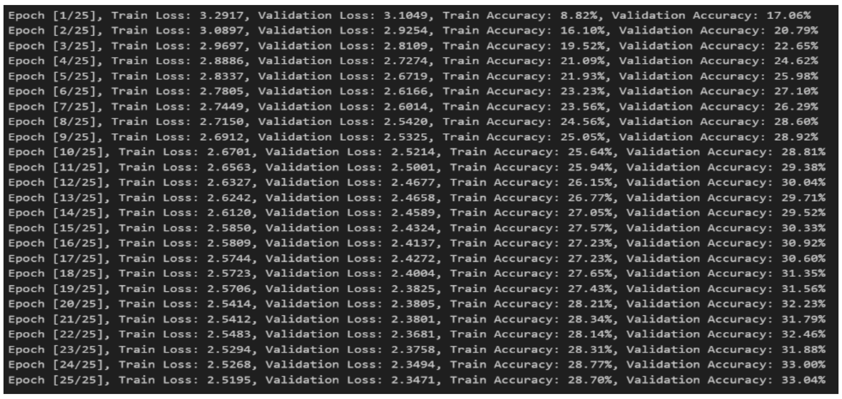
This change led to better results, where now we have got 75.57% test accuracy, among all methods and architectures we tried, this was the best accuracy increase we have achieved. In our opinion this increase could be the result of adding variability by cropping, which makes the model focus on different features in the pictures instead of specific features of the country, helping the model generalize better to unseen data.

Weight Freezing:

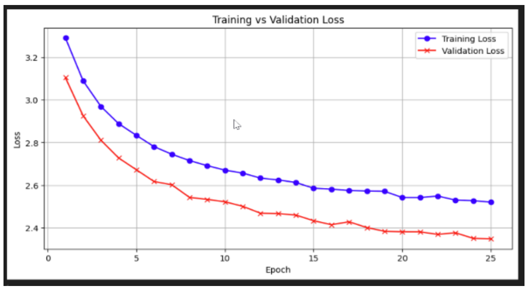
[5] [6] [7] The Freeze Unfreeze method is a trick used for pretrained models like ResNet50, with this method, we initially freeze the earlier layers of the model during training, which prevents their weights from being affected from the backpropagation phase, and only train the last layer on our dataset for few epochs, and after that, we unfreeze the other layers and train the whole model on our data for more epochs, this leads to the model focusing on the features learned from the pretrained data at first, while training the last layer on our dataset.

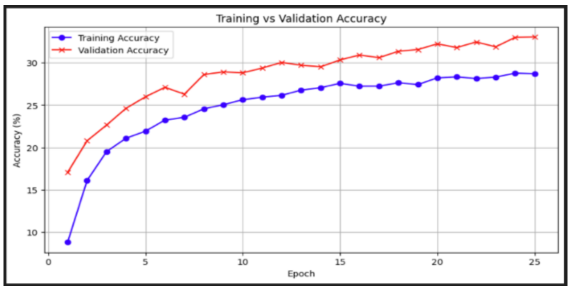
Results:

Initially, we froze all layers except the last layer and trained for a few epochs, these are the results for the first part of training:

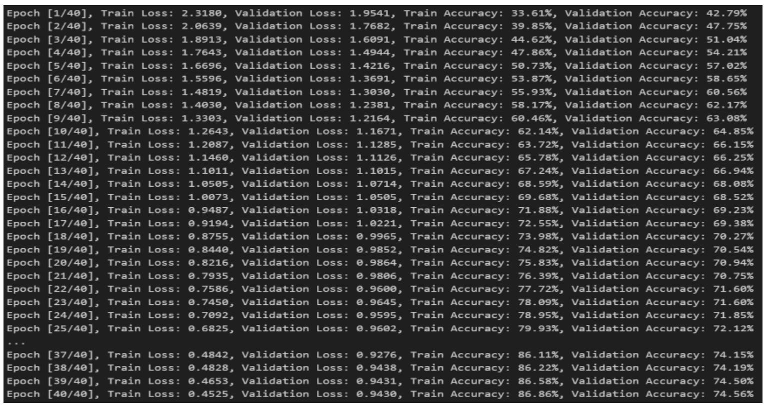


```
Epoch [1/25], Train Loss: 3.2917, Validation Loss: 3.1049, Train Accuracy: 8.82%, Validation Accuracy: 17.06%
Epoch [2/25], Train Loss: 3.0897, Validation Loss: 2.9254, Train Accuracy: 16.10%, Validation Accuracy: 20.79%
Epoch [3/25], Train Loss: 2.9697, Validation Loss: 2.8109, Train Accuracy: 19.52%, Validation Accuracy: 22.65%
Epoch [4/25], Train Loss: 2.8886, Validation Loss: 2.7274, Train Accuracy: 21.09%, Validation Accuracy: 24.62%
Epoch [5/25], Train Loss: 2.8337, Validation Loss: 2.6719, Train Accuracy: 21.93%, Validation Accuracy: 25.98%
Epoch [6/25], Train Loss: 2.7805, Validation Loss: 2.6166, Train Accuracy: 23.23%, Validation Accuracy: 27.10%
Epoch [7/25], Train Loss: 2.7449, Validation Loss: 2.6014, Train Accuracy: 23.56%, Validation Accuracy: 26.29%
Epoch [8/25], Train Loss: 2.7150, Validation Loss: 2.5420, Train Accuracy: 24.56%, Validation Accuracy: 28.60%
Epoch [9/25], Train Loss: 2.6912, Validation Loss: 2.5325, Train Accuracy: 25.05%, Validation Accuracy: 28.92%
Epoch [10/25], Train Loss: 2.6701, Validation Loss: 2.5214, Train Accuracy: 25.64%, Validation Accuracy: 28.81%
Epoch [11/25], Train Loss: 2.6563, Validation Loss: 2.5001, Train Accuracy: 25.94%, Validation Accuracy: 29.38%
Epoch [12/25], Train Loss: 2.6327, Validation Loss: 2.4677, Train Accuracy: 26.15%, Validation Accuracy: 30.04%
Epoch [13/25], Train Loss: 2.6242, Validation Loss: 2.4658, Train Accuracy: 26.77%, Validation Accuracy: 29.71%
Epoch [14/25], Train Loss: 2.6120, Validation Loss: 2.4589, Train Accuracy: 27.05%, Validation Accuracy: 29.52%
Epoch [15/25], Train Loss: 2.5850, Validation Loss: 2.4324, Train Accuracy: 27.57%, Validation Accuracy: 30.33%
Epoch [16/25], Train Loss: 2.5809, Validation Loss: 2.4137, Train Accuracy: 27.23%, Validation Accuracy: 30.92%
Epoch [17/25], Train Loss: 2.5744, Validation Loss: 2.4272, Train Accuracy: 27.23%, Validation Accuracy: 30.60%
Epoch [18/25], Train Loss: 2.5723, Validation Loss: 2.4004, Train Accuracy: 27.65%, Validation Accuracy: 31.35%
Epoch [19/25], Train Loss: 2.5706, Validation Loss: 2.3825, Train Accuracy: 27.43%, Validation Accuracy: 31.56%
Epoch [20/25], Train Loss: 2.5414, Validation Loss: 2.3805, Train Accuracy: 28.21%, Validation Accuracy: 32.23%
Epoch [21/25], Train Loss: 2.5412, Validation Loss: 2.3801, Train Accuracy: 28.34%, Validation Accuracy: 31.79%
Epoch [22/25], Train Loss: 2.5483, Validation Loss: 2.3681, Train Accuracy: 28.14%, Validation Accuracy: 32.46%
Epoch [23/25], Train Loss: 2.5294, Validation Loss: 2.3758, Train Accuracy: 28.31%, Validation Accuracy: 31.88%
Epoch [24/25], Train Loss: 2.5268, Validation Loss: 2.3494, Train Accuracy: 28.77%, Validation Accuracy: 33.00%
Epoch [25/25], Train Loss: 2.5195, Validation Loss: 2.3471, Train Accuracy: 28.70%, Validation Accuracy: 33.04%
```

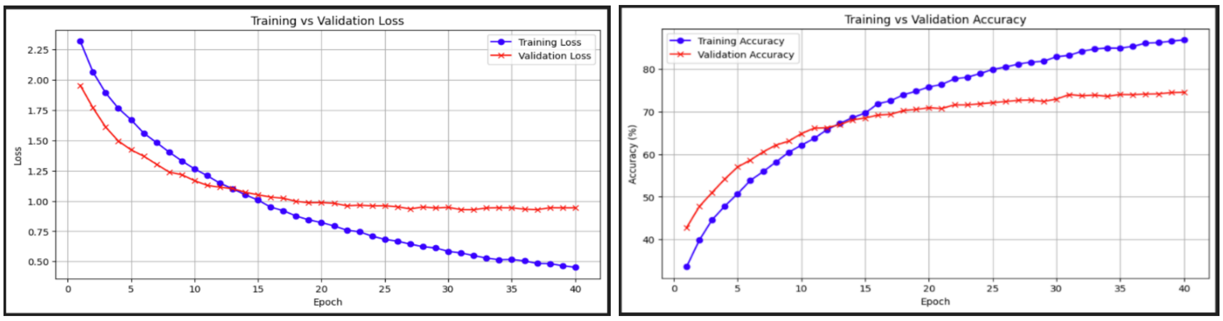




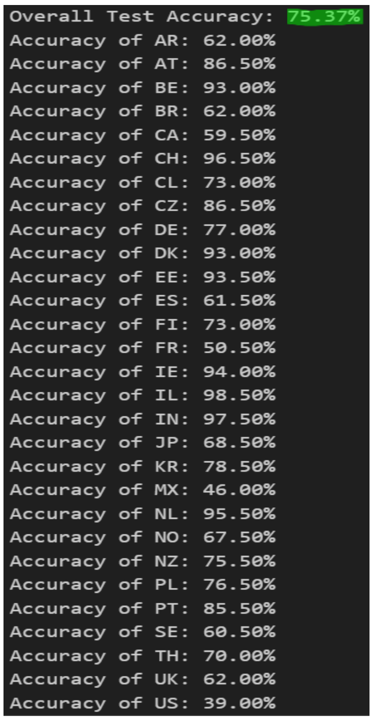
After unfreezing all layers and training the whole model for more epochs:



```
Epoch [1/40], Train Loss: 2.3180, Validation Loss: 1.9541, Train Accuracy: 33.61%, Validation Accuracy: 42.79%
Epoch [2/40], Train Loss: 2.0639, Validation Loss: 1.7682, Train Accuracy: 39.85%, Validation Accuracy: 47.75%
Epoch [3/40], Train Loss: 1.8913, Validation Loss: 1.6091, Train Accuracy: 44.62%, Validation Accuracy: 51.04%
Epoch [4/40], Train Loss: 1.7643, Validation Loss: 1.4944, Train Accuracy: 47.86%, Validation Accuracy: 54.21%
Epoch [5/40], Train Loss: 1.6696, Validation Loss: 1.4216, Train Accuracy: 50.73%, Validation Accuracy: 57.02%
Epoch [6/40], Train Loss: 1.5596, Validation Loss: 1.3691, Train Accuracy: 53.87%, Validation Accuracy: 58.65%
Epoch [7/40], Train Loss: 1.4819, Validation Loss: 1.3030, Train Accuracy: 55.93%, Validation Accuracy: 60.56%
Epoch [8/40], Train Loss: 1.4030, Validation Loss: 1.2381, Train Accuracy: 58.17%, Validation Accuracy: 62.17%
Epoch [9/40], Train Loss: 1.3303, Validation Loss: 1.2164, Train Accuracy: 60.46%, Validation Accuracy: 63.08%
Epoch [10/40], Train Loss: 1.2643, Validation Loss: 1.1671, Train Accuracy: 62.14%, Validation Accuracy: 64.85%
Epoch [11/40], Train Loss: 1.2087, Validation Loss: 1.1285, Train Accuracy: 63.72%, Validation Accuracy: 66.15%
Epoch [12/40], Train Loss: 1.1460, Validation Loss: 1.1126, Train Accuracy: 65.78%, Validation Accuracy: 66.25%
Epoch [13/40], Train Loss: 1.1011, Validation Loss: 1.1015, Train Accuracy: 67.24%, Validation Accuracy: 66.94%
Epoch [14/40], Train Loss: 1.0505, Validation Loss: 1.0714, Train Accuracy: 68.59%, Validation Accuracy: 68.08%
Epoch [15/40], Train Loss: 1.0073, Validation Loss: 1.0505, Train Accuracy: 69.68%, Validation Accuracy: 68.52%
Epoch [16/40], Train Loss: 0.9487, Validation Loss: 1.0318, Train Accuracy: 71.88%, Validation Accuracy: 69.23%
Epoch [17/40], Train Loss: 0.9194, Validation Loss: 1.0221, Train Accuracy: 72.55%, Validation Accuracy: 69.38%
Epoch [18/40], Train Loss: 0.8755, Validation Loss: 0.9965, Train Accuracy: 73.98%, Validation Accuracy: 70.27%
Epoch [19/40], Train Loss: 0.8440, Validation Loss: 0.9852, Train Accuracy: 74.82%, Validation Accuracy: 70.54%
Epoch [20/40], Train Loss: 0.8216, Validation Loss: 0.9864, Train Accuracy: 75.83%, Validation Accuracy: 70.94%
Epoch [21/40], Train Loss: 0.7935, Validation Loss: 0.9806, Train Accuracy: 76.39%, Validation Accuracy: 70.75%
Epoch [22/40], Train Loss: 0.7586, Validation Loss: 0.9600, Train Accuracy: 77.72%, Validation Accuracy: 71.60%
Epoch [23/40], Train Loss: 0.7450, Validation Loss: 0.9645, Train Accuracy: 78.09%, Validation Accuracy: 71.60%
Epoch [24/40], Train Loss: 0.7092, Validation Loss: 0.9595, Train Accuracy: 78.95%, Validation Accuracy: 71.85%
Epoch [25/40], Train Loss: 0.6825, Validation Loss: 0.9602, Train Accuracy: 79.93%, Validation Accuracy: 72.12%
...
Epoch [37/40], Train Loss: 0.4842, Validation Loss: 0.9276, Train Accuracy: 86.11%, Validation Accuracy: 74.15%
Epoch [38/40], Train Loss: 0.4828, Validation Loss: 0.9438, Train Accuracy: 86.22%, Validation Accuracy: 74.19%
Epoch [39/40], Train Loss: 0.4653, Validation Loss: 0.9431, Train Accuracy: 86.58%, Validation Accuracy: 74.50%
Epoch [40/40], Train Loss: 0.4525, Validation Loss: 0.9430, Train Accuracy: 86.86%, Validation Accuracy: 74.56%
```



Test accuracy:

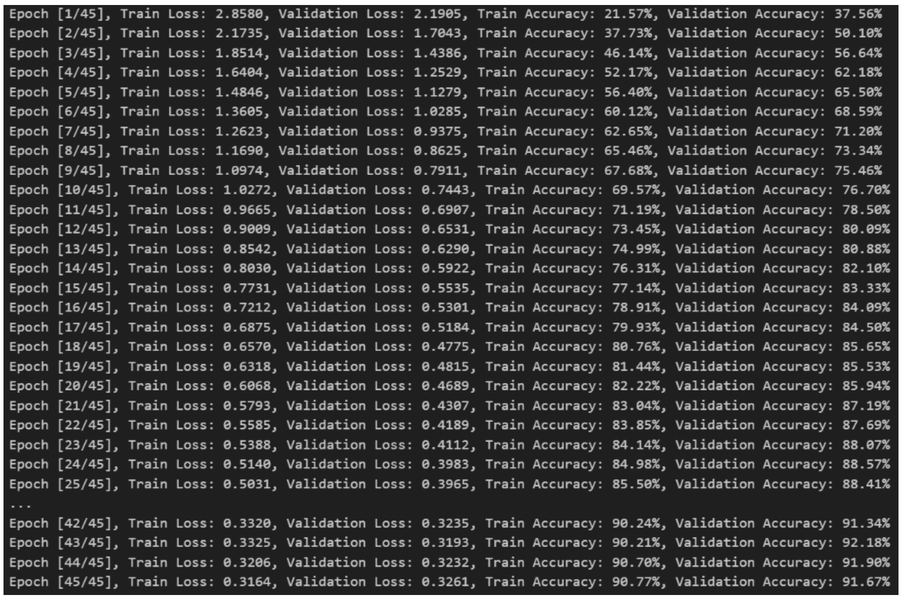


```
Overall Test Accuracy: 75.37%
Accuracy of AR: 62.00%
Accuracy of AT: 86.50%
Accuracy of BE: 93.00%
Accuracy of BR: 62.00%
Accuracy of CA: 59.50%
Accuracy of CH: 96.50%
Accuracy of CL: 73.00%
Accuracy of CZ: 86.50%
Accuracy of DE: 77.00%
Accuracy of DK: 93.00%
Accuracy of EE: 93.50%
Accuracy of ES: 61.50%
Accuracy of FI: 73.00%
Accuracy of FR: 50.50%
Accuracy of IE: 94.00%
Accuracy of IL: 98.50%
Accuracy of IN: 97.50%
Accuracy of JP: 68.50%
Accuracy of KR: 78.50%
Accuracy of MX: 46.00%
Accuracy of NL: 95.50%
Accuracy of NO: 67.50%
Accuracy of NZ: 75.50%
Accuracy of PL: 76.50%
Accuracy of PT: 85.50%
Accuracy of SE: 60.50%
Accuracy of TH: 70.00%
Accuracy of UK: 62.00%
Accuracy of US: 39.00%
```

Despite not getting a clear improvement in the test accuracy compared to the random crop experiment, we have noticed that using this method led to countries like France Mexico and USA that had low accuracies compared to other countries, to be improved by a decent margin. This increase could be the result of the strength of pretrained models like ResNet50, which are optimized for general features in such pictures like edges or shapes, and despite the fact that the pictures that ResNet50 the fact that ResNet50 was originally trained on a different dataset with different characteristics, this method worked because in the first few epochs, we made sure that the pretrained features remained intact and didn’t get affected from training, allowing the model to use these strong general features effectively. By freezing the early layers, the model concentrated on learning the specific features needed to tell apart countries like France, Mexico, and the USA. When we unfroze the layers later, the model made small adjustments to its earlier layers, helping it better handle the unique features in our dataset while keeping the important knowledge from its original training.

Data augmentation:
Like we mentioned before, data collection was very time consuming, therefore we looked for a way to artificially inflate our data size without compromising its quality, so we tried adding the mirror images, we took each image flipped it horizontally and added it into the dataset, this way we would have 2000 images per country to work with.

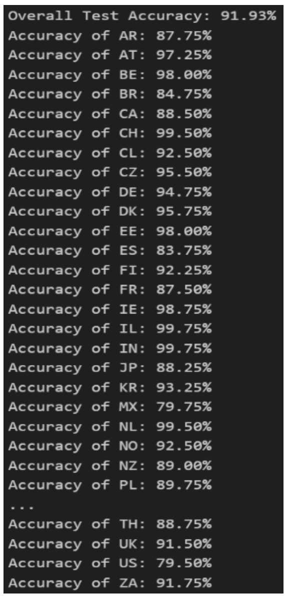
Results:



```
Epoch [1/45], Train Loss: 2.8580, Validation Loss: 2.1905, Train Accuracy: 21.57%, Validation Accuracy: 37.56%
Epoch [2/45], Train Loss: 2.1735, Validation Loss: 1.7043, Train Accuracy: 37.73%, Validation Accuracy: 50.10%
Epoch [3/45], Train Loss: 1.8514, Validation Loss: 1.4386, Train Accuracy: 46.14%, Validation Accuracy: 56.64%
Epoch [4/45], Train Loss: 1.6404, Validation Loss: 1.2529, Train Accuracy: 52.17%, Validation Accuracy: 62.18%
Epoch [5/45], Train Loss: 1.4846, Validation Loss: 1.1279, Train Accuracy: 56.40%, Validation Accuracy: 65.50%
Epoch [6/45], Train Loss: 1.3605, Validation Loss: 1.0285, Train Accuracy: 60.12%, Validation Accuracy: 68.59%
Epoch [7/45], Train Loss: 1.2623, Validation Loss: 0.9375, Train Accuracy: 62.65%, Validation Accuracy: 71.20%
Epoch [8/45], Train Loss: 1.1690, Validation Loss: 0.8625, Train Accuracy: 65.46%, Validation Accuracy: 73.34%
Epoch [9/45], Train Loss: 1.0974, Validation Loss: 0.7911, Train Accuracy: 67.68%, Validation Accuracy: 75.46%
Epoch [10/45], Train Loss: 1.0272, Validation Loss: 0.7443, Train Accuracy: 69.57%, Validation Accuracy: 76.70%
Epoch [11/45], Train Loss: 0.9665, Validation Loss: 0.6907, Train Accuracy: 71.19%, Validation Accuracy: 78.50%
Epoch [12/45], Train Loss: 0.9009, Validation Loss: 0.6531, Train Accuracy: 73.45%, Validation Accuracy: 80.09%
Epoch [13/45], Train Loss: 0.8542, Validation Loss: 0.6290, Train Accuracy: 74.99%, Validation Accuracy: 80.88%
Epoch [14/45], Train Loss: 0.8030, Validation Loss: 0.5922, Train Accuracy: 76.31%, Validation Accuracy: 82.10%
Epoch [15/45], Train Loss: 0.7731, Validation Loss: 0.5535, Train Accuracy: 77.14%, Validation Accuracy: 83.33%
Epoch [16/45], Train Loss: 0.7212, Validation Loss: 0.5301, Train Accuracy: 78.91%, Validation Accuracy: 84.09%
Epoch [17/45], Train Loss: 0.6875, Validation Loss: 0.5184, Train Accuracy: 79.93%, Validation Accuracy: 84.50%
Epoch [18/45], Train Loss: 0.6570, Validation Loss: 0.4775, Train Accuracy: 80.76%, Validation Accuracy: 85.65%
Epoch [19/45], Train Loss: 0.6318, Validation Loss: 0.4815, Train Accuracy: 81.44%, Validation Accuracy: 85.53%
Epoch [20/45], Train Loss: 0.6068, Validation Loss: 0.4689, Train Accuracy: 82.22%, Validation Accuracy: 85.94%
Epoch [21/45], Train Loss: 0.5793, Validation Loss: 0.4307, Train Accuracy: 83.04%, Validation Accuracy: 87.19%
Epoch [22/45], Train Loss: 0.5585, Validation Loss: 0.4189, Train Accuracy: 83.85%, Validation Accuracy: 87.69%
Epoch [23/45], Train Loss: 0.5388, Validation Loss: 0.4112, Train Accuracy: 84.14%, Validation Accuracy: 88.07%
Epoch [24/45], Train Loss: 0.5140, Validation Loss: 0.3983, Train Accuracy: 84.98%, Validation Accuracy: 88.57%
Epoch [25/45], Train Loss: 0.5031, Validation Loss: 0.3965, Train Accuracy: 85.50%, Validation Accuracy: 88.41%
...
Epoch [42/45], Train Loss: 0.3320, Validation Loss: 0.3235, Train Accuracy: 90.24%, Validation Accuracy: 91.34%
Epoch [43/45], Train Loss: 0.3325, Validation Loss: 0.3193, Train Accuracy: 90.21%, Validation Accuracy: 92.18%
Epoch [44/45], Train Loss: 0.3206, Validation Loss: 0.3232, Train Accuracy: 90.70%, Validation Accuracy: 91.90%
Epoch [45/45], Train Loss: 0.3164, Validation Loss: 0.3261, Train Accuracy: 90.77%, Validation Accuracy: 91.67%
```

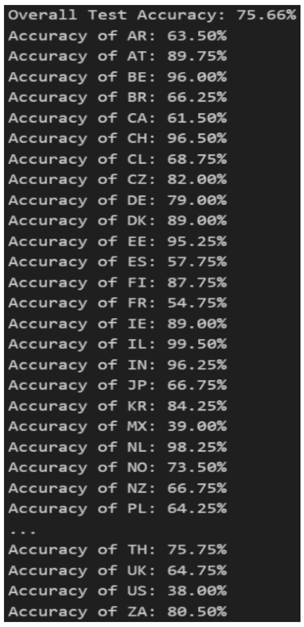
At first we were very happy with these results, however, on second glance, they seemed strange, we have never seen the validation accuracy grow larger than the training accuracy, not just while working on this project, but in previous experiences, so we though this might be cause by training contamination, or as we later discovered it to be called data leakage, where a picture appears in the train set and its mirror appears in the validation or test set, making this data considered to be “seen” data, defying the definition of validation set and test set, both being composed of “unseen” data points, to confirm this we reran the experiment but now we made sure that an image and its reflection where in the same sets.

Before:



```
Overall Test Accuracy: 91.93%
Accuracy of AR: 87.75%
Accuracy of AT: 97.25%
Accuracy of BE: 98.00%
Accuracy of BR: 84.75%
Accuracy of CA: 88.50%
Accuracy of CH: 99.50%
Accuracy of CL: 92.50%
Accuracy of CZ: 95.50%
Accuracy of DE: 94.75%
Accuracy of DK: 95.75%
Accuracy of EE: 98.00%
Accuracy of ES: 83.75%
Accuracy of FI: 92.25%
Accuracy of FR: 87.50%
Accuracy of IE: 98.75%
Accuracy of IL: 99.75%
Accuracy of IN: 99.75%
Accuracy of JP: 88.25%
Accuracy of KR: 93.25%
Accuracy of MX: 79.75%
Accuracy of NL: 99.50%
Accuracy of NO: 92.50%
Accuracy of NZ: 89.00%
Accuracy of PL: 89.75%
...
Accuracy of TH: 88.75%
Accuracy of UK: 91.50%
Accuracy of US: 79.50%
Accuracy of ZA: 91.75%
```

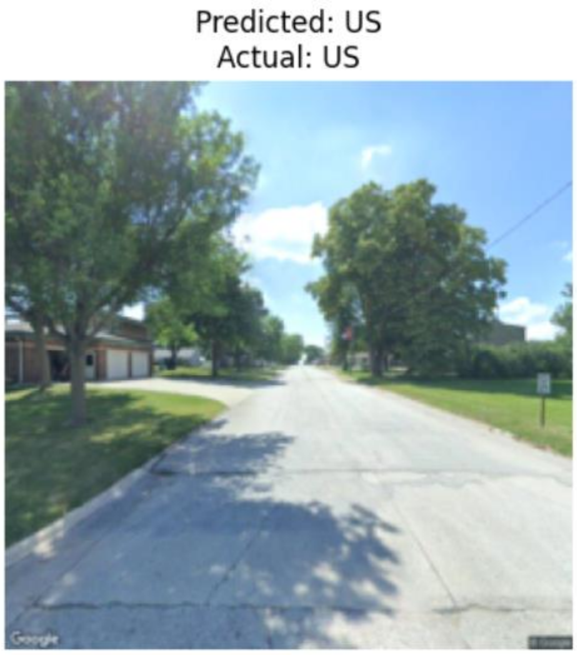
After:

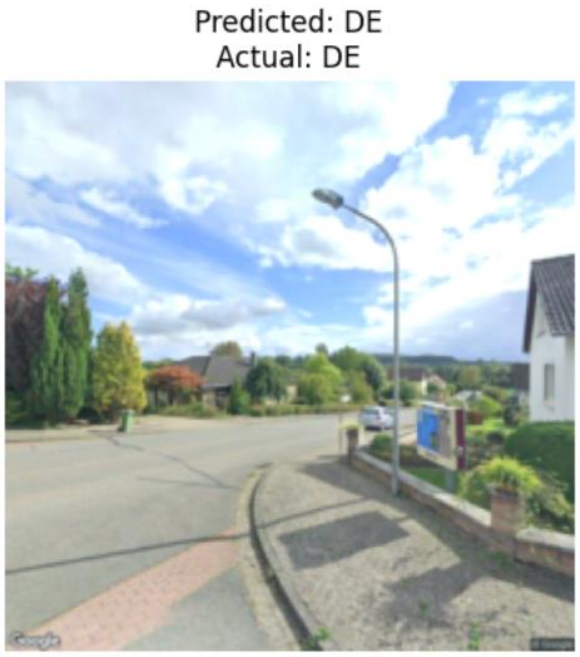


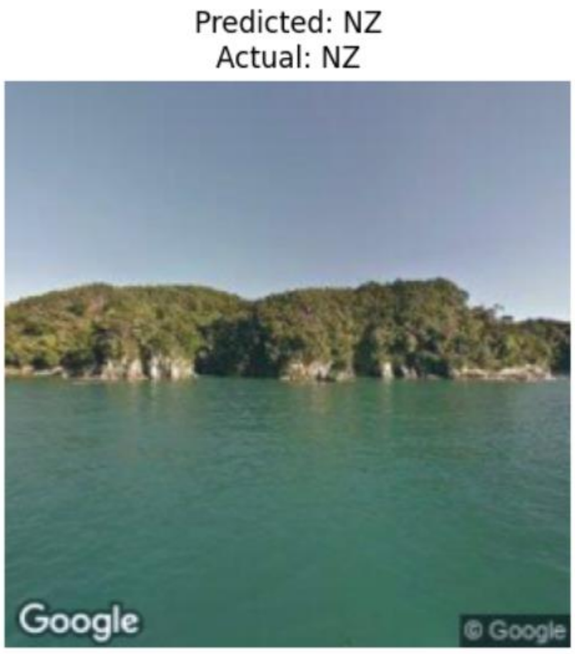
```
Overall Test Accuracy: 75.66%
Accuracy of AR: 63.50%
Accuracy of AT: 89.75%
Accuracy of BE: 96.00%
Accuracy of BR: 66.25%
Accuracy of CA: 61.50%
Accuracy of CH: 96.50%
Accuracy of CL: 68.75%
Accuracy of CZ: 82.00%
Accuracy of DE: 79.00%
Accuracy of DK: 89.00%
Accuracy of EE: 95.25%
Accuracy of ES: 57.75%
Accuracy of FI: 87.75%
Accuracy of FR: 54.75%
Accuracy of IE: 89.00%
Accuracy of IL: 99.50%
Accuracy of IN: 96.25%
Accuracy of JP: 66.75%
Accuracy of KR: 84.25%
Accuracy of MX: 39.00%
Accuracy of NL: 98.25%
Accuracy of NO: 73.50%
Accuracy of NZ: 66.75%
Accuracy of PL: 64.25%
...
Accuracy of TH: 75.75%
Accuracy of UK: 64.75%
Accuracy of US: 38.00%
Accuracy of ZA: 80.50%
```

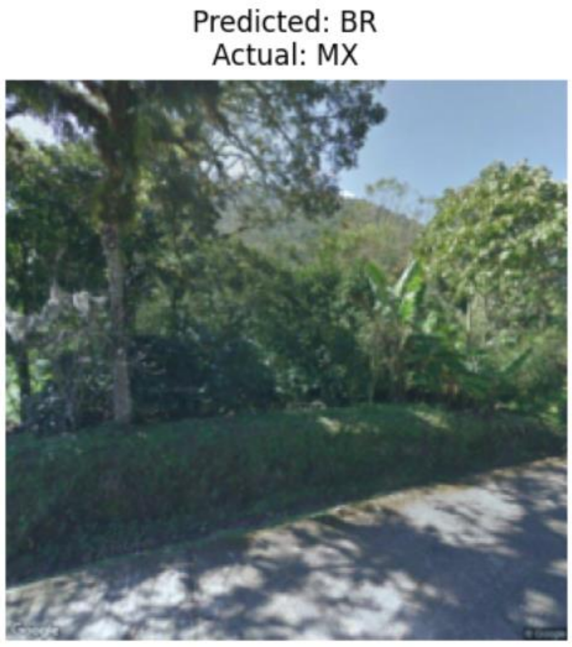
As we can see, when we made sure that there is no data leakage our test accuracy came back down, confirming we had data leakage, we did see an improvement of a percentage and a half which is better than nothing.

Model predictions (ResNet-50 with cropping):









Final thoughts:

Given the complexity of the problem, we are pleased with a final accuracy of 75%, It was surprising to see that even the simple models we used where able to achieve an accuracy of 36%, we would have guessed it to be much lower, maybe with more data and some hyper-tunning these models would have done even better, another surprise was the similarity between the MobileNet and ResNet-50 models, with MobileNet over preforming and reaching such a high accuracy, we would have predicted it to outperform the basic models but not to compete with ResNet.
It was interesting to see what countries the model struggled with and which countries it shined on, overall, we got an impressively capable model relatively to our data size, as similar models developed by others and shared on the internet achieved better results but had much larger dataset sizes, with some having 10000 pictures per country and reaching an accuracy of 95% with different architectures.
A major challenge we encountered while working on this project was the data collection, it is a surprisingly hard task as country boarders are complex and collecting the pictures took a lot of time, insuring the quality of the data we collected was very important and took many iterations to get right.

Direction for Future Research:

1. As we mentioned many times, a major concern with the methodology was the data size, it would be interesting to collect more data and gradually train the different models with more and more data to see how could this affect the accuracy of the models, a good experiment would have us starting with, for instance, 4000 pictures per country, splitting into train validation and test sets and train the ResNet model on an increasing number of pictures, starting with 200, 400, 800 … we would predict that the more data the better but it would be interesting to see at what number do we get diminishing returns, if such point exists.
2. It would also be interesting to try different architectures for this task, while deciding what models we wanted to use we had to leave some out, such as tinyViT, which is known for its efficiency and generalization capability [8].
3. Throughout the project we conducted 3 experiment which saw some improvement in the test accuracy of the ResNet-50 model compared to its benchmark, these being the weight freezing, data augmentation and cropping, each one of these occur in a different phase of the training process, meaning they don’t conflict, so they could all be combined to train a single model which could theoretically benefit from each technique, it would be interesting to test this hypothesis and determine the extent of the improvement.
4. Finally, it would be interesting to expand the problem to all available countries, how would these models preform on double the number of countries we used?

Use of LLMs:

In this project, we aimed to minimize the use of ChatGPT or other large language models (LLMs) to ensure that our submission reflects our own efforts, creativity, and hard work. We spent a long time and effort implementing the project independently. However, there were specific cases where we used ChatGPT to save some time on specific small tasks or when we faced challenges in deciding how to approach certain problems. These instances are as follows:

- ChatGPT was used to help with the code for displaying the accuracy grids shown above. Specifically, it assisted in suggesting methods to structure and display the grids properly.
- We asked for help in producing the code to display pictures from the dataset with their actual and predicted labels. This feature which is included in the final sections of the Python files, was an interesting addition we wanted to implement but were unsure on how to display images in a Jupyter notebook.
- ChatGPT provided insights and tips about ResNet50 and MobileNetV2 that were not available in the resources referenced in our bibliography.
- Some parts of the final report were checked for grammatical accuracy using ChatGPT to ensure clarity.
- Used ChatGPT for clarification on how to use Google’s API.
- Used ChatGPT to calculate the input and output channel sizes in the custom CNN model’s convolution layers.
- Used ChatGPT to declare the TransformedDataset class which we used to apply different transformations on our data sets.
- Used to proof read and some rephrasing.

Bibliography:

[1] https://keras.io/

[2] https://arxiv.org/

[3] https://paperswithcode.com/

[4] https://blog.roboflow.com/

[5] https://medium.com/

[6] https://stackoverflow.com/

[7] https://www.quora.com/

[8] https://arxiv.org/abs/2207.10666