Neocis Assessment Report

Karanveer Singh

November 14, 2022

1 How to run the code:

1.1 inference.py

One of the main deliverable of the project is a file that performs inference on a given test dataset and outputs the results in a expected format, this functionality is impletmented in the inference.py file and this can be executed using the following command:

```
python3 inference.py -m <path to the saved model>
-n <name of the model: mobilenet/convnext>
-d <path to the test dataset> -i <path to the index to class mapping file>
```

Alternatively if all the required files are placed in the same directory as the inference.py file and then the default arguments are set in place to run inference on the mobilenet model and can be executed using the following command:

```
python3 inference.py
```

1.2 video_to_frames.py

This file extract the frames from a all the videos and makes a train folder and saves the frames into folders with their respective class name, fine details of the implementation are explained in detail in the comments and documentation of the code inside the file, this file can be executed using the following command:

```
python3 video_to_frames.py -p <path to the folder containing the videos>
```

Alternatively if the folder containing the video is in the same directory as this file is then the following command can be used to execute this file:

```
python3 video_to_frames.py
```

1.3 val_split.py

This file randomly samples 20 percent of the files from each class and moves them to the validation folder with same file structure as the train folder, fine details of the implementation are explained in detail in the comments and documentation of the code inside the file, this file can be executed using the following command:

```
python3 val_split.py -p <path to the train folder>
```

Alternatively if the train folder is in the same directory as this file is then the following command can be used to execute this file:

```
python3 val_split.py
```

1.4 test_folder.py

This file essentially just copied all the frames from the validation folder to the test folder but does not retain the class names all the frames from all the classes are placed in a single folder called test, fine details of the implementation are explained in detail in the comments and documentation of the code inside the file, this file can be executed using the following command:

```
python3 test_folder.py -p <path to the validation folder>
```

Alternatively if the validation folder is in the same directory as this file is then the following command can be used to execute this file:

```
python3 test_folder.py
```

1.5 convnext.ipynb and mobilenet.ipynb

These files were used for model development and training pipeline and can be executed like any other iPython notebooks, I chose iPython Notebooks for this because it makes prototyping, experimenting and debugging faster and more efficinet for me, a lot of the fine details and documentation for the training procedure can be found in the markdowns, comments, function signatures and documentations inside the notebooks.

2 Introduction

Key new concepts: Image classification , pattern recognition, convolutional neural networks.

Implementation: Implemented CNN-based architectures including MobileNet and ConvNeXt.

2.1 Executive Summary

The key to this project is the pattern recognition problems that require *position invariance*. Specifically I worked on the problem of recognizing the size of drill bits in images. In typical pictures of drill bits, the drill bits are rarely perfectly centered. Different pictures of the same drill bit may have the bits shifted by varying amounts. The classifier must recognize the drill bits regardless of this indeterminacy of position. This calls for position-invariant models, specifically Convolutional Neural Networks, or CNNs.

A CNN is a neural network that derives representations (or embeddings) of input images that are expected to be invariant to the precise positions of the patterns in it. These embeddings are subsequently classified by downstream classifiers (which may just be an additional softmax layer, or even an MLP, appended after the convolutional layers), to achieve position-invariant pattern recognition.

In the first, *classification*, we will attempt to identify the size of the drill bit in a picture. This is a *closed set* problem, where the drill bits in the test set have also been seen in the training set, although the precise pictures in the test set will not be in the training set.

3 Project Goal

The goal of this project is to build an end-to-end system the extracts frames containing drill bits from videos and then classifies the drill bits based on the size, the objective is to attain as hight accuracy as possible while also keeping the model size reasonably small.

4 Methodology

4.1 Residual Networks

Having a network that is good at feature extraction and being able to efficiently train that network is the core of the classification task. This requires to train very deep neural networks and as it turns out deep neural networks are difficult to train because they suffer from vanishing and exploding gradients types of problems, here I will use skip connections that allow us to take the activations of one layer and suddenly feed it to another layer even much deeper in the network, using that I can build residual networks (resnets) which enable us to train very deep neural networks, sometimes even networks of over a hundred layers.

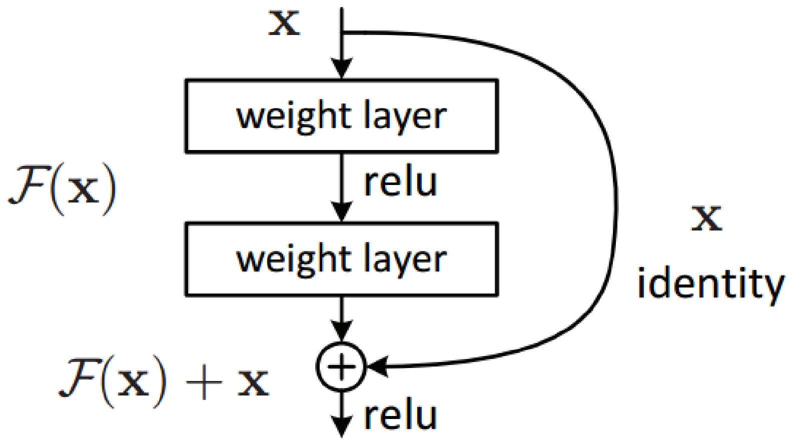

Figure 1: A Residual Block

Resnets are made of something called residual blocks, which are a set of layers that are connected to each other, and the input of the first layer is added to the output of the last layer in the block. This is called a residual connection. This identity mapping does not have any parameters and is just there to add the input to the output of the block. This allows deeper networks to be built and trained efficiently.

4.1.1 ResNet

ResNet models were proposed in "Deep Residual Learning for Image Recognition"[1]. Here we have the 5 versions of ResNet models, which contain 18, 34, 50, 101, and 152 layers, respectively. Detailed model architectures can be found in the paper linked above.

4.1.2 ConvNeXt Tiny

ConvNeXt is a very recently proposed CNN architecture that uses inverted bottlenecks inspired by the Swin Transformer, residual blocks, and depthwise separable convolutions instead of regular convolutions. I wanted to try this model because it supposedly beats the state-of-the-art Vision Transformers in image classification task and so I wanted to test it out, it achieved very high validation accuracy of 94.7 percent but it had close to 27 million parameters so I wanted to try a light weight model next, the details of the ConvNeXt Architecture can be found in the following paper"A ConvNet for the 2020s".[2]

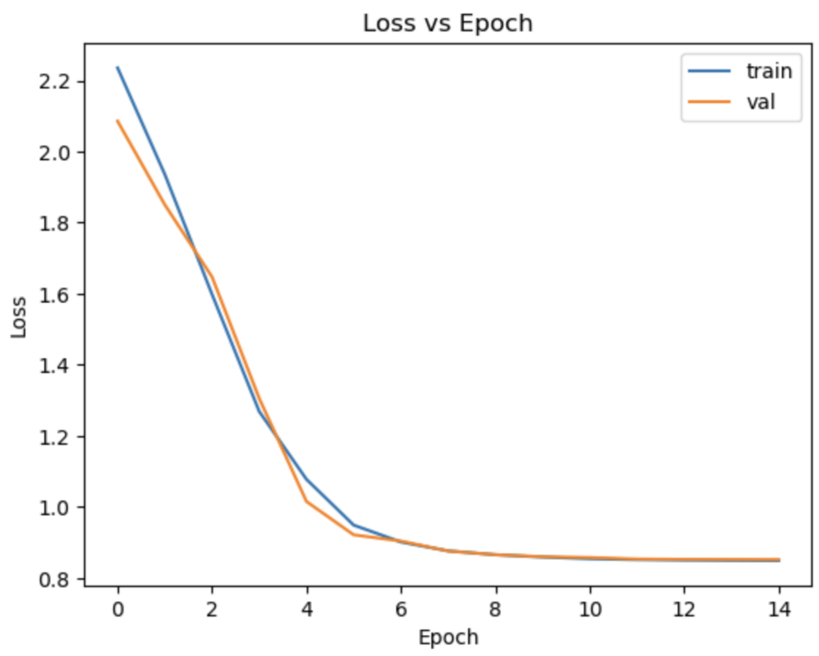

Figure 2: Loss vs Epoch for ConvNext

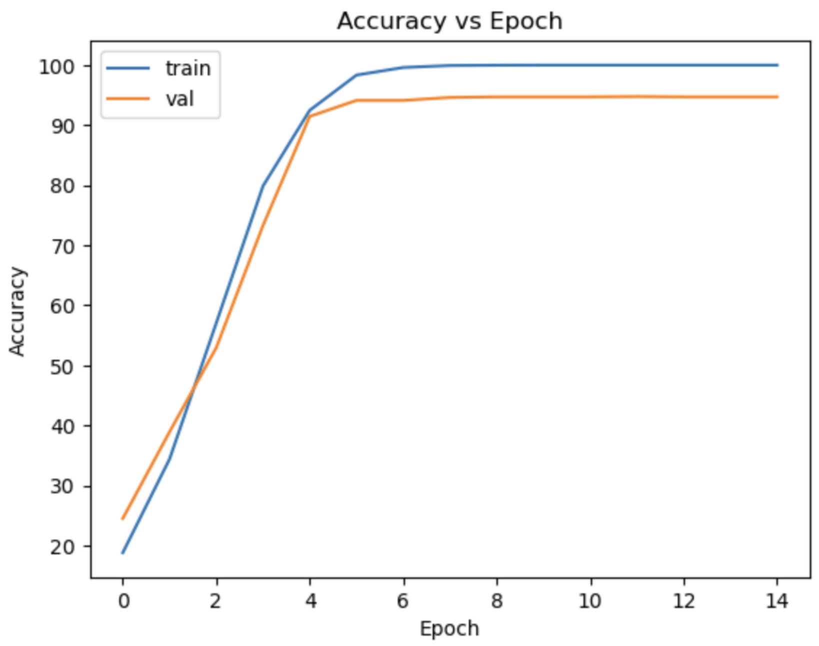

Figure 3: Accuracy vs Epoch for ConvNext

4.1.3 MobileNet V3 Small

This is a much smaller model but efficient model, and it was also able to achieve similar performance as ConvNeXt but with much fewer parameters (less then 1 million parameters), which lead me to the conclusion that we don't need a bigger model to perform well on this task, I imported model from torchvision with imagenet weights and fine tuned the model by further training it on our dataset with cross-entropy loss. The detals of the architecture can be found in the paper cited.[3]

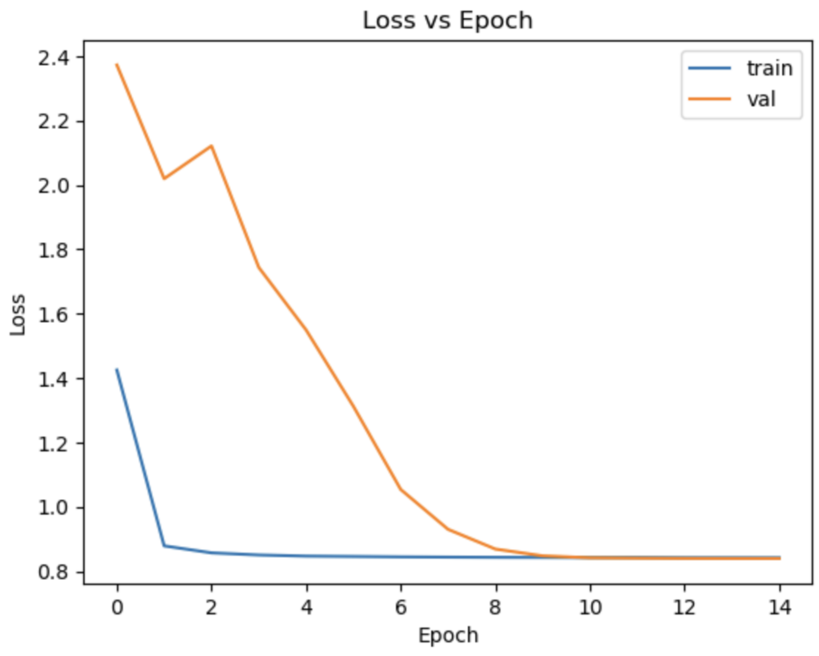

Figure 4: Loss vs Epoch for MobileNet

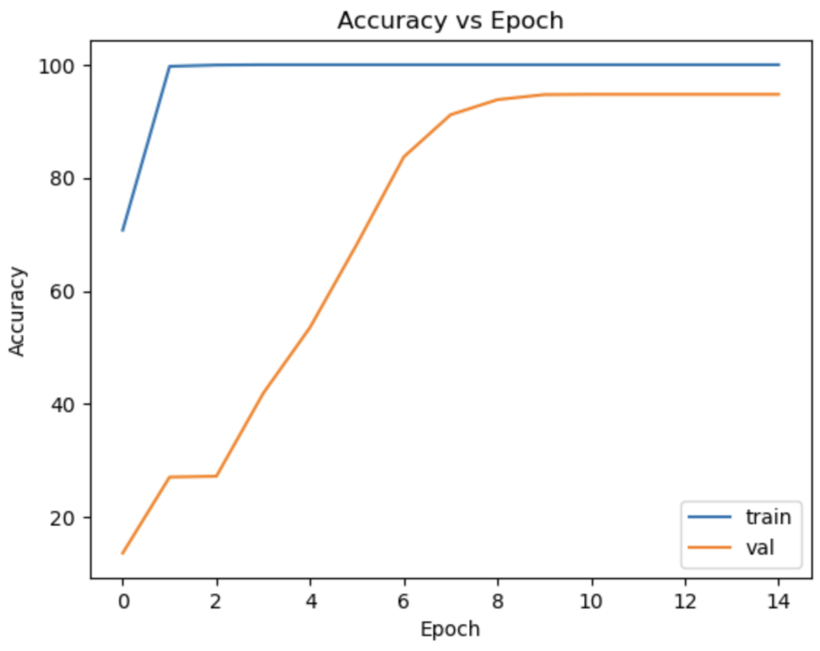

Figure 5: Accuracy vs Epoch for MobileNet

4.2 Loss Function:cross-entropy loss

The natural choice for the loss function is cross-entropy loss because this is a multi class classification task.

5 Dataset

The data was created by extracting individual frames from the videos that were provided and was kept in 3 sparate directores:

1. The first one is the train folder which contains 80 percent of the images each of which is stored in a subfolder of the name of the class it belongs to
2. The second one is the validation folder which has identical file structure as the train folder but has the other 80 percent of the images used for validation
3. All the images in the validation folder are copied into the test folder but are not segregated according to class, it did not make sense to make a new test set because all the images are coming from the same distribution(the videos)

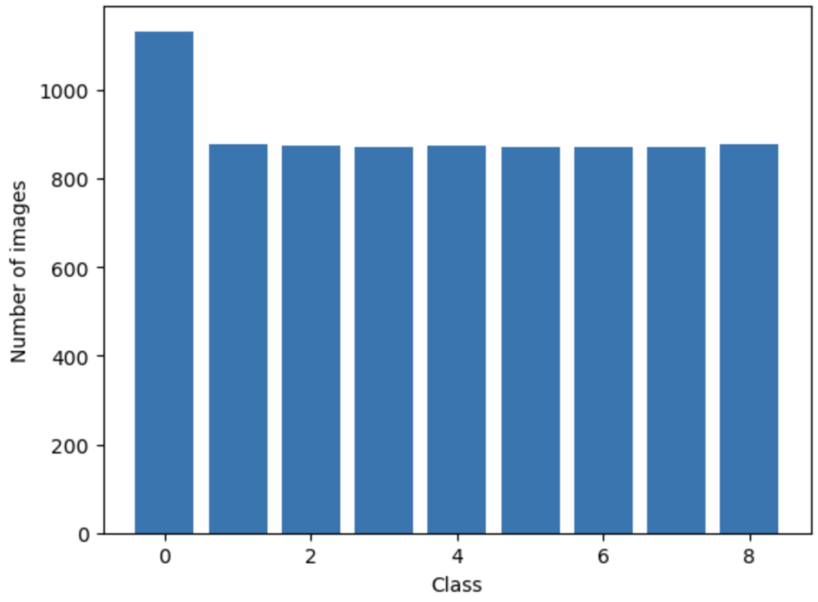

Figure 6: Data distribution by class

This distribution shows that the data is fairly class balanced and so we don't have to worry much about class imbalance

6 Discussion

1. Both the models performed almost identically but the MobileNet model has much fewer parameters and is more parameter efficient for this task, so if we were to deploy this I would recommend deploying the MobileNet model.
2. I also tried to ensemble the results from both the models and though there was a slight improvement in accuracy, it is very insignificant which leads me to believe that both the models have learned similar feature extraction.
3. These models are very robust in terms of patterns recognition and are translation invariant but are not rotation invariant, I also did not do Random Rotation augmentation to bring any kind of Rotation in variance because I assumed the images in test will also be taken from a camera mounted identically to the videos that were provided.
4. This method could give unreliable results if the images that are fed into the network have any kind of pre-processing that I have not trained it for in my training data.

5. Finally the distribution of the test images by predicted class as seen in the figure below matched almost identically with the train data, which makes sense because test data is a copy of the validation data and the validation data was created by randomly sampling 20 percent of the training data and thus this leads me to believe that the model is performing pretty well and had generalised on the validation data.

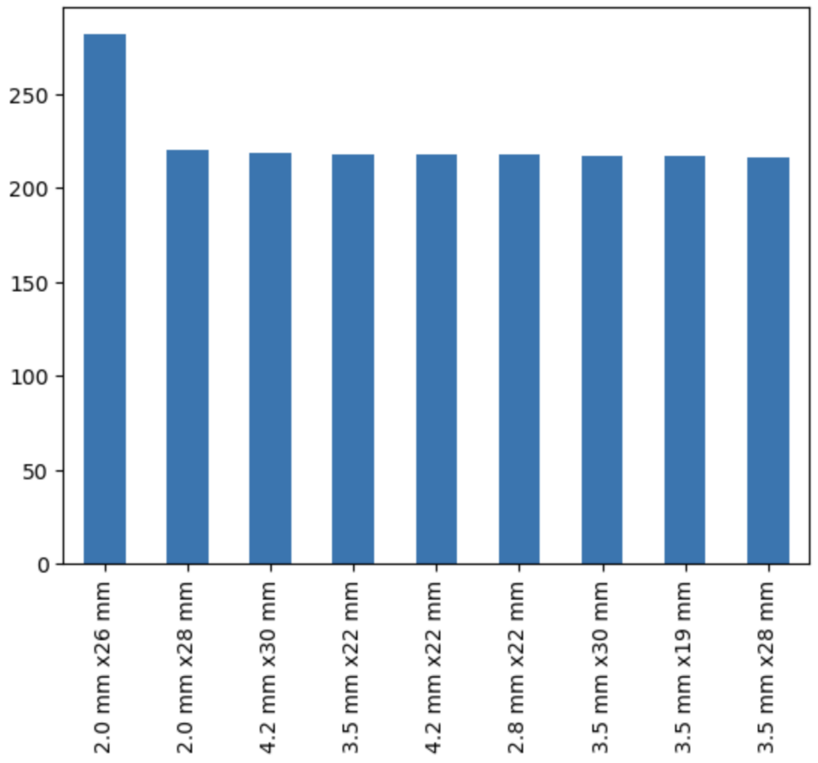

Figure 7: Predicted distribution by class

References

[1] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[2] Zhuang Liu et al. *A ConvNet for the 2020s*. 2022. DOI: 10.48550/ARXIV.2201.03545. URL: https://arxiv.org/abs/2201.03545.

[3] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. cite arxiv:1704.04861. 2017. URL: http://arxiv.org/abs/1704.04861.