Freudenberg Defect Leather Classification Use Case

Jonathan Calderón Mora

September 2024

1. Introduction

The main objective of this use case is to evaluate the author Jonathan Calderón Mora on his technical knowledge.

For all industries giving a high end product is important for its reputation and trust in the market. The leather industry is no less, however after all the manufacturing process defects can appear on the leather skin. Manual inspection of all the products it's subjective to the eye and a tedious and time-consuming responsibility. The solution is a system that could identify defects on the leather and even classify it .

To automatize the process of classifying locate leather defects we will use machine learning to help us recognize the different defects on the leather such as folding marks, growth marks, grain off, loose grain, and pinholes.

The solution proposed will be deep learning neural network that will use computer vision to localize and classify said defects.

2. Problem Statement

The problem on hand is that the only way to identify damage or defects on a piece of leather we need a trained eye and we can only check a equal number of pieces of leather as eyes trained we have, plus it has an extra time cost just to find the defect, if any, and then classify it.

Like it has been explained before with this technology we can organize an assembling line fully automatized that recognize the defect, and depending on the defect or if it's a good piece it can be sorted into continuing in the manufacturing process, save it for later use if the damage is not serious or to remove it completely form the process.

The goals in this project will be, find the best setup for time and accuracy efficiency to train our AI to recognize damaged leather.

This time we will be using Python and Tensorflow Keras to create different models and training them. We will be using the models offered by the library and we will be using a Convolutional Neuronal Network or CNN made manually, a Resnet50 and finally a Visual Geometry Group or VGG.

```python
def create_cnn_model(inputShape, numClasses, activationNotDense):
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation=activationNotDense, input_shape=inputShape))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation=activationNotDense))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(128, (3, 3), activation=activationNotDense))
    model.add(layers.Flatten())
    model.add(layers.Dense(128, activation=activationNotDense))
    model.add(layers.Dense(numClasses, activation='softmax'))
    return model

def create_resnet_model(inputShape, numClasses, activationNotDense):
    base_model = tf.keras.applications.ResNet50(input_shape=inputShape, include_top=False, weights='imagenet')
    base_model.trainable = False

    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(numClasses, activation='softmax')
    ])
    return model

def create_vgg_model(inputShape, numClasses, activationNotDense):
    model = models.Sequential()
    model.add(layers.Conv2D(64, (3, 3), activation=activationNotDense, input_shape=inputShape))
    model.add(layers.Conv2D(64, (3, 3), activation=activationNotDense))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(128, (3, 3), activation=activationNotDense))
    model.add(layers.Conv2D(128, (3, 3), activation=activationNotDense))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(256, (3, 3), activation=activationNotDense))
    model.add(layers.Conv2D(256, (3, 3), activation=activationNotDense))
    model.add(layers.Conv2D(256, (3, 3), activation=activationNotDense))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(512, activation=activationNotDense))
    model.add(layers.Dense(numClasses, activation='softmax'))
    return model
```

Figura 1: Code Snippets of the Models Creation

3. Machine Learning Component

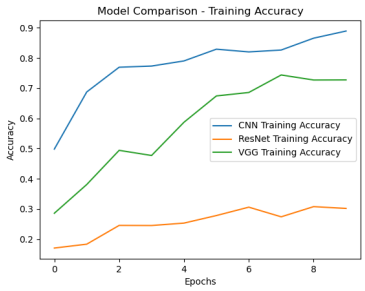
As explained before the models used will be a CNN created by ours, the Restnet50 model given by Keras and a VGG. The algorithms of activation will be the relu or the switch (selu) and the activation algorithm for the dense layer will always be 'softmax'.

For the data collection we are using the defect leather public dataset by Kaggle which contains 3600 images, they will be normalized and escalated to a size of 224x224 pixels for all models but the principal reason for that is that the VGG need images to be 224x224 pixels.

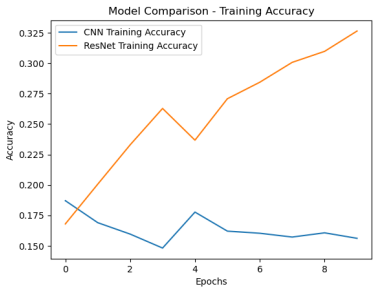
All the models are working under 10 epochs in the same machine, using 'adam' as an optimizer and we are testing time for epoch and accuracy.

3.1. Results and Performance Metrics

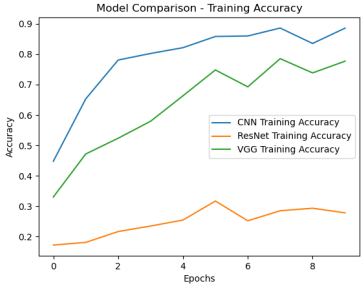
The results are the next:



(a) 32 batch ReLu



(b) 32 batch SeLu



(c) 64 batch ReLu

Figura 2: Epoch to Accuracy relation

As we can observe the 'relu' did a much better performance than 'selu' and we don't have data for the VGG on 'selu' because of a hardware issue.

When working with the 'relu' configuration our CNN can achieve almost a 90 % of accuracy it seems that we top that with bigger batches of data, the VGG doesn't seem to learn at the same rate as our own CNN but it gives nice results on accuracy, roaming around 70 %.

On the other side, when working with a 'selu' configuration we can see really bad results of accuracies around 30 % for the VGG and going down in every epoch for our CNN. All that worse accuracy can be explain by the internal component and working of 'selu', we didn't accommodate the data to what 'selu' and its internal normalization need.

Lastly a note on the VGG model, he did equally bad on all the models, but it not going down on the 'selu' configuration and looking like it can continue

ramping for some more epochs make me think that because that model was created with an internal function of the Keras library maybe we can make an approach for the 'selu' config only working with the models Keras offers.

4. Case Study or Example

Machine Learning in leather working is a really recent topic only been worked on for a decade, all the investigation and dataset usually are private. That means that there is market to explode and problems to resolve, some university in Asian countries like Indonesia are working on different models that can recognize if the leather is synthetic or comes from an animal and can even tell which animal.

5. Challenges and Solutions

Some of the challenges seen on this case were:

- Tensorflow not working on Pyhton 3.12, a downgrade was needed to Python 3.10
- VGG not working with a 'selu' configuration, probably due to a hardware limitation
- 'selu' configuration giving a much worse accuracy, 'selu' needs to be studied with more time to understand his internal normalization which usually mess with all the previous data management.

6. Future Work and Improvements

For the models, a big improvement will be make 'selu' work so we can play with another learning algorithm that recently has been praised by the scientific community. We can try to make our own models work with it or working with the internal models of Keras, just to see if we can make it work and study more about the algorithm in question.

To improve this use case, we could focus one type of defect that we think it will more usual on our manufacturing process so we can avoid it with an almost 100 % accuracy.

For a future work and with a better dataset I will like to work on a Leather Classification system, where from the texture of the leather with it can recognize if it's synthetic or if it's animal, and if it's animal then which animal come from, as told before.

7. Conclusion

Starting this project I could never thought about all the process the leather goes by, how all the steps can make a dent on the skin or how different the manufacturing process can be depending on the origin of the leather.

Also, I think we have nice results with two of the models giving an accuracy above 70 % with light work and one of them even almost making it to 90 %.