



CNN : Image Classifier for Vegetables

Wilfred Djumin

DAAA/FT/2B/05



Overview

BACKGROUND INFO

EDA & LOADING DATA

FEATURE ENGINEERING

MODELLING

+

MODEL IMPROVEMENT

+

EVALUATION

EVALUATION & SUMMARY

Background Info

- Our task is to implement a **multi-class image classifier** , using **CNN models**
- Given Color images of **224 by 224 pixels**, containing **15 types** of vegetables (class)
- We are to build **2 models**, one for input sizes **31 by 31**, another for input sizes **128 by 128**



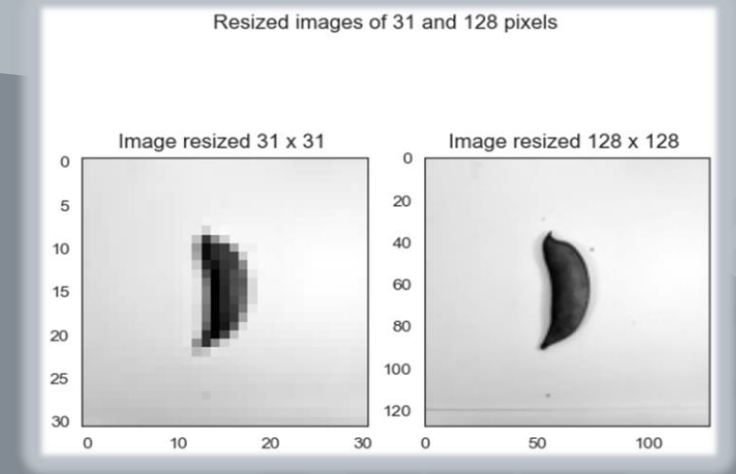
The background features a light blue base with large, organic, overlapping shapes in muted olive green and dusty rose. A stylized pine branch is visible in the upper left corner. Two thin, white, curved lines sweep across the bottom right. A thin horizontal line is positioned near the bottom of the frame.

EDA & Loading Data

Data Loading & EDA

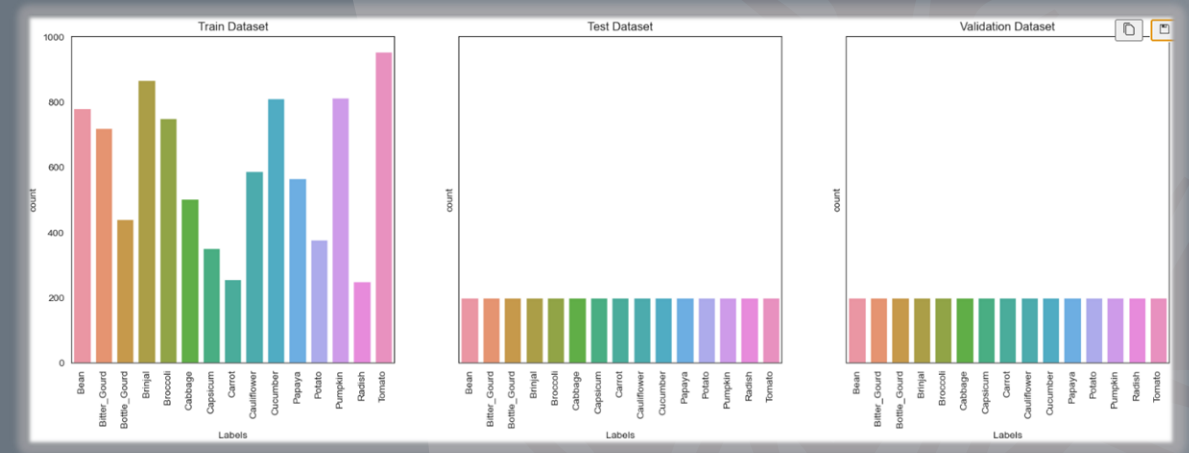
DATA LOADING METHODS

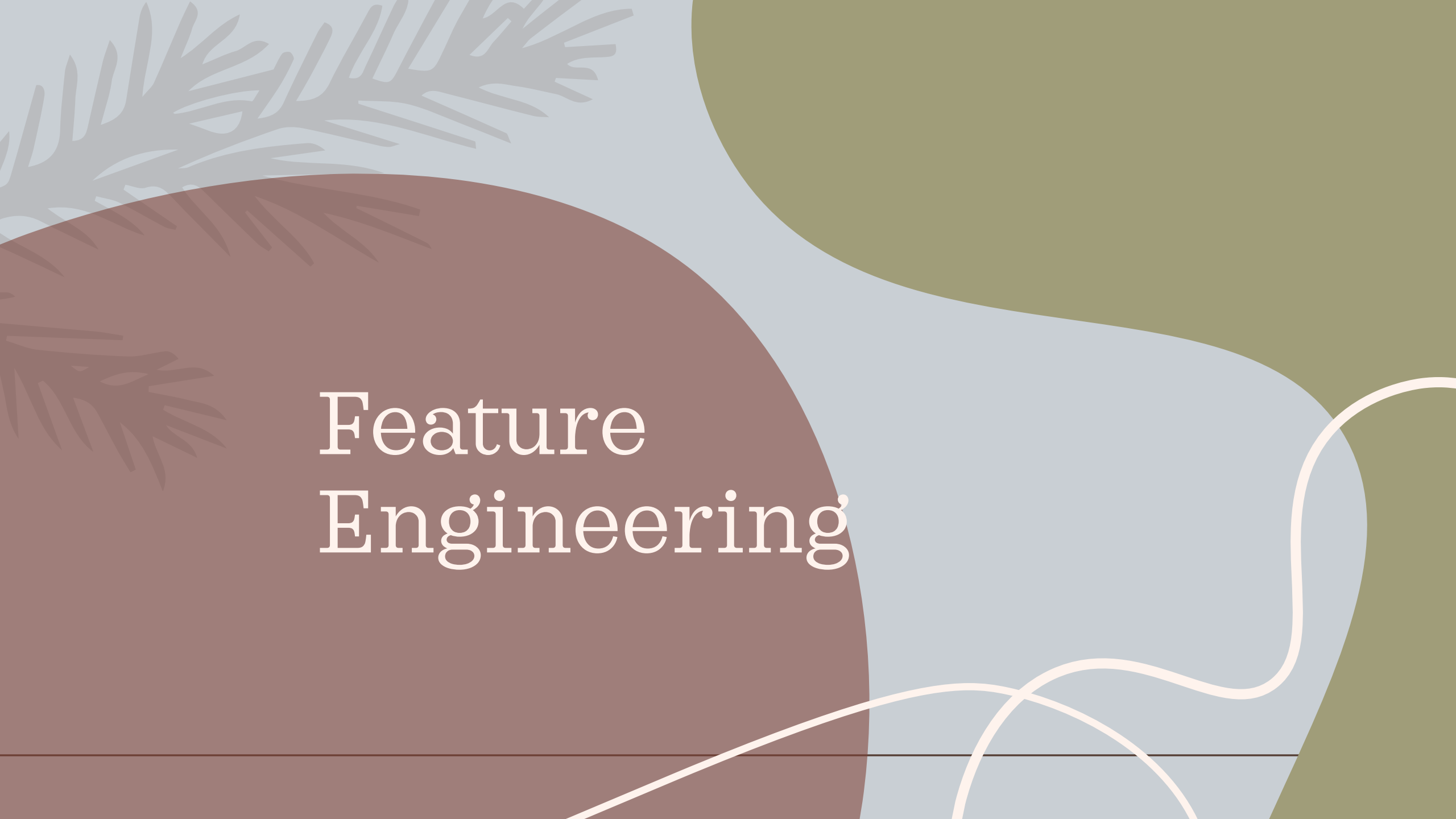
- Main transformations applied to the images are :
 - Convert them to grayscale
 - Resizing to $31 * 31$ & $128 * 128$
 - Normalizing pixels between 0 and 1



EXPLORATORY DATA ANALYSIS

- Looking at Class Distribution counts
 - Train set contained class imbalance, could impact model performance
 - We should try to address this using methods such as SMOTE, Oversampling, or using class weights which we will touch on later



The background features a light blue base with large, organic, overlapping shapes in muted olive green and a dusty rose color. In the top left corner, there is a stylized, light grey pine branch. Two thin, white, wavy lines curve across the bottom right portion of the image.

Feature Engineering

Dealing with Class Imbalance

Main Methods that we explored:

- **SMOTE** - generates synthetic samples using interpolation between existing minority samples
- **Oversampling** - replicates existing minority samples to balance the dataset.
- **Class Weights Initialization** - Estimate class weights for unbalanced datasets.

Results:

- Both oversampling and SMOTE gave us **new train size** of **14325** compared to **9028** earlier
- For **class weights**, minority classes like **Pumpkin** were given a **greater weight**, while majority classes were given less weights

Data Augmentation

- Based on our dataset, we have ~**9000 train images** , **3000 validation** and **test** images, which can be argued to be **quite limited**, given that **each class** would contain **+ -600 images** on average
- I tried applying **augmentation techniques** like:
 - **Width and Height shifting** (0.2)
 - **Random Image rotation** between -45 to 45 degrees
 - **Random Zoom range** from 75% to 125%
- Used **ImageDataGenerator** as well as **flow_from_directory** for augmentation, which helps create images on the fly (during training)
- Additionally, I tried applying **Augmentation** on top of the **Oversampled Train set** & **SMOTE** on **Train set**

The background features a light blue base with large, organic, overlapping shapes in muted olive green and a dusty rose color. In the top left corner, there is a stylized, light grey pine branch. Two thin, white, wavy lines curve across the bottom right portion of the image.

Modelling

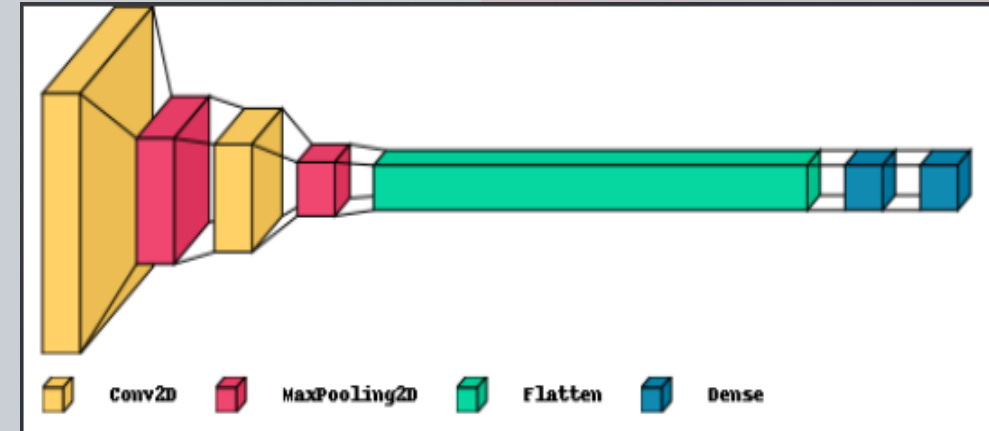
Baseline Models

Model Architecture (for both input sizes)

- Conv2D + MaxPooling2D (32 nodes)
- Conv2D + MaxPooling2D (64 nodes)
- Flatten
- Dense layer with 128 nodes
- Final Dense / Output layer (15 nodes)
- Activation – ReLU
- Output Activation - SoftMax

Feeding Model With (both input sizes):

- Original Train set
- Train set + SMOTE
- Train set + oversampling
- Trainset + class weights

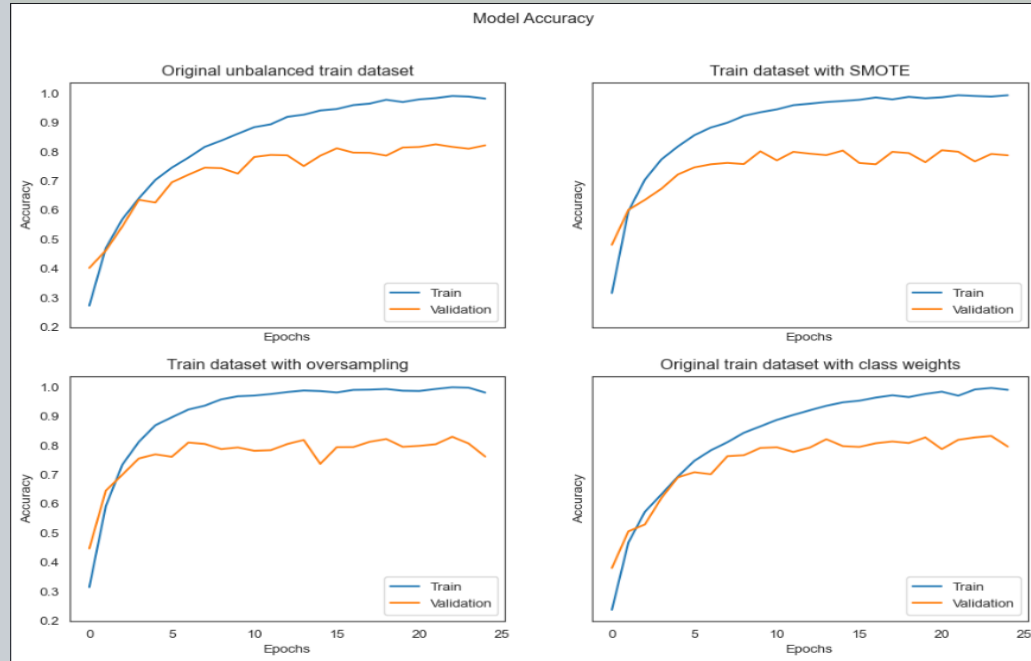


```
=====
Total params: 315,791
Trainable params: 315,791
Non-trainable params: 0
...
Total params: 7,393,679
Trainable params: 7,393,679
Non-trainable params: 0
=====
```

Model Params for Input B is about 23 x more than for Input A , which would mean more computationally intense / longer training times for Input B

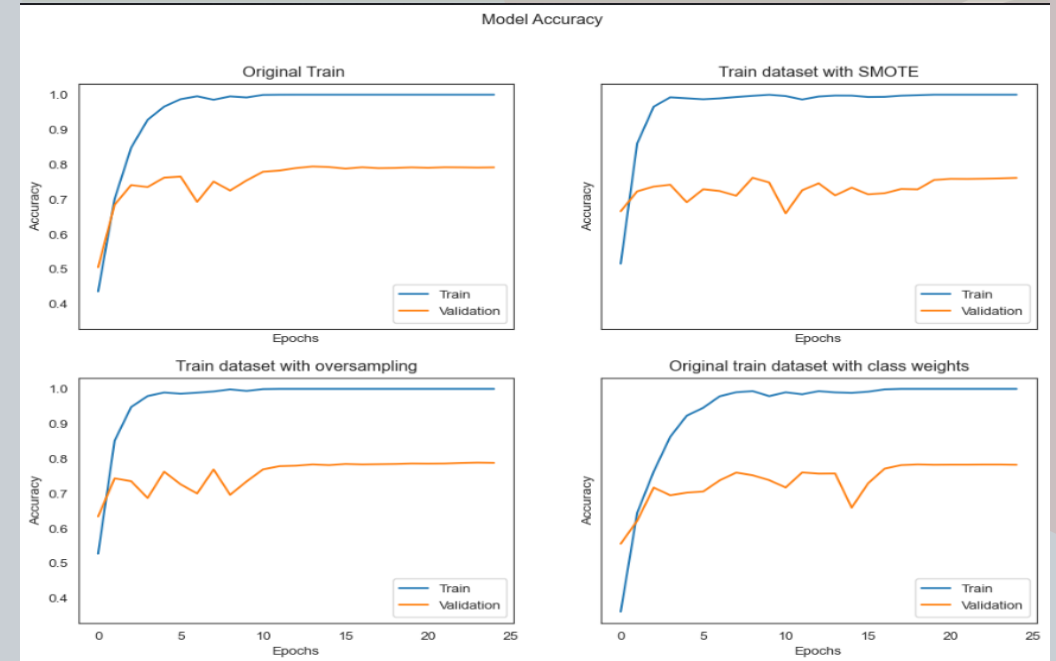
Baseline Models Results

Input Size 31 by 31



- In general, the Model **overfits** in all 4 configurations.
- Class weights seems to work better compared to other techniques, in this scenario
- **0.82 Accuracy on test set** for Model trained with **original train set**

Input Size 128 by 128



- In general, the Model **overfits** significantly as seen from the **huge gap** between the train and validation accuracies
- **0.77 Accuracy on test set** for Model trained with original train set

Reducing Overfit + Adding Layers

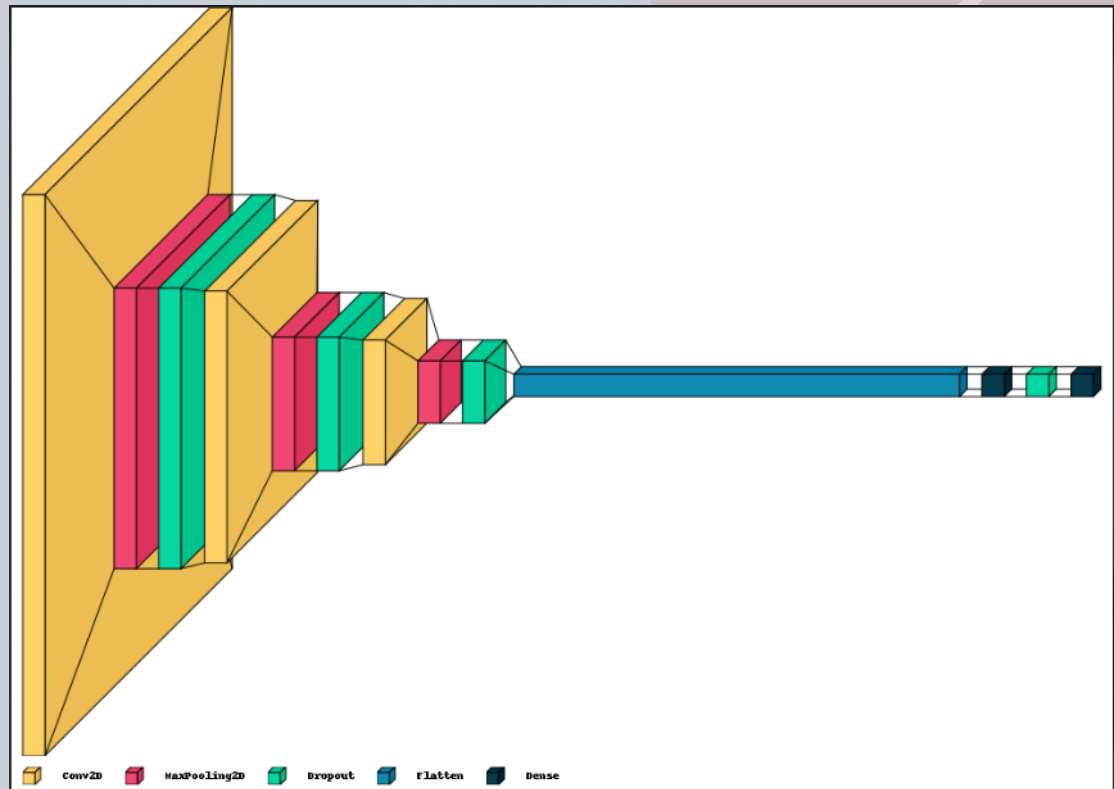
Model Architecture (for both input sizes)

- 3 * Conv2D & MaxPooling2D (32,64,128 nodes)
- Dropout (0.3) between each layer(s)
- Dense layer with 128 nodes
- Final Dense / Output layer (15 nodes)
- Activation – ReLU
- Output Activation - SoftMax

Feeding Model With (both input sizes):

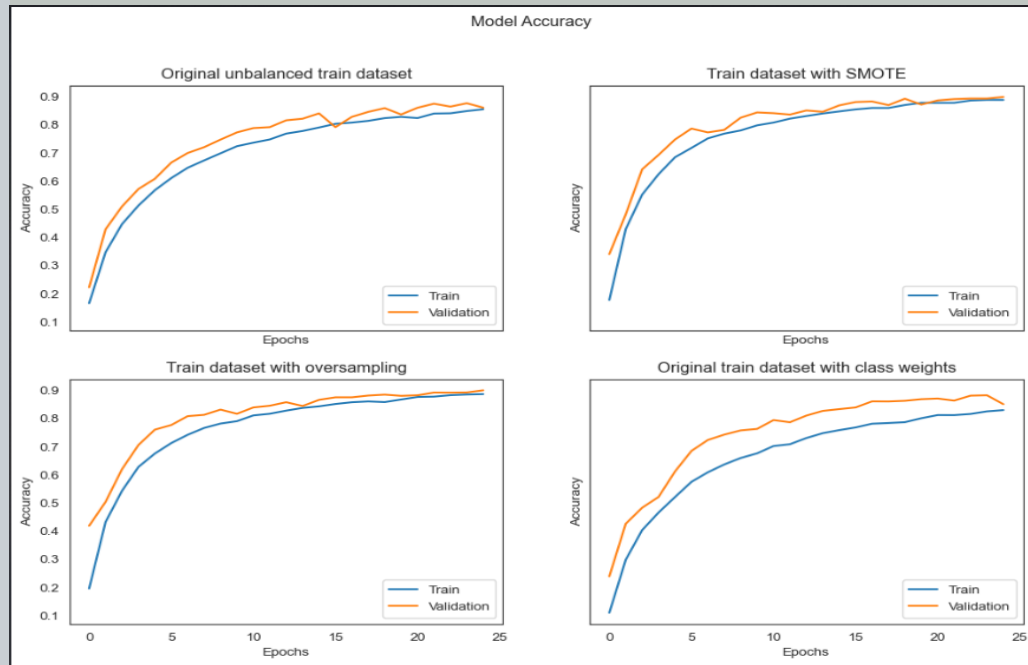
- Original Train set
- Train set + **SMOTE**
- Train set + **oversampling**
- Trainset + **class weights**

Visual Representation of Model Architecture



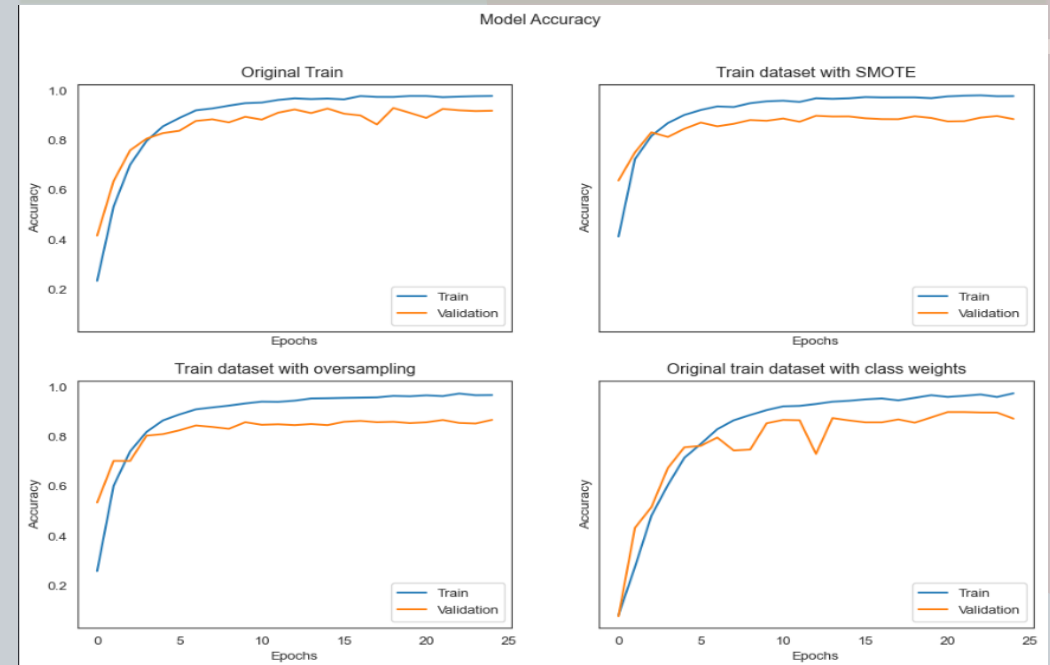
Baseline Models With Improvements

Input Size 31 by 31



- Model seems to be **overfitting less** than before which is mostly due to the **dropout layers** being introduced
- Both SMOTE & Oversampling indeed allowed model to **learn faster** while **not overfitting**
- **0.89 Accuracy** on **test set** for Model trained with **original train set**

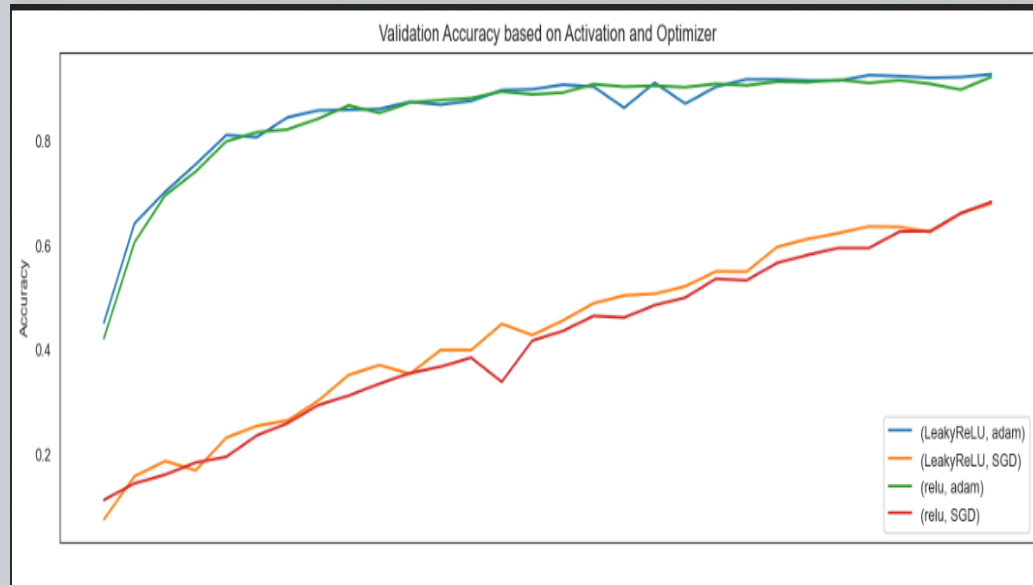
Input Size 128 by 128



- In general, the Model **still overfits** in all 4 configurations.
- It's quite interesting to note that the Model seems to have similar/better performance without **class imbalance techniques**
- **0.93 Accuracy** on **test set** for Model trained with **original train set**

Trying Different Activations & Optimizers

Input Size 31 by 31 (Oversampled)



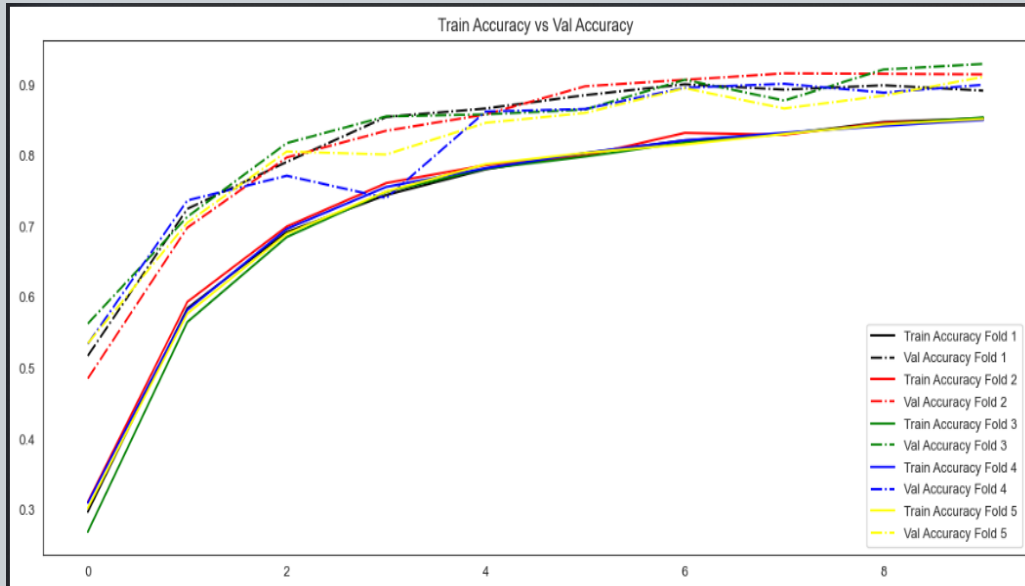
Input Size 128 by 128 (Oversampled)



- Model(similar architecture to improved baseline) was fed with **Oversampled Train set for both Inputs**
- Both **ReLU** and **LeakyReLU** has similar performances (expected)
- **Adam** seems to work better than **SGD**
- We would proceed with using **LeakyReLU** and **Adam**

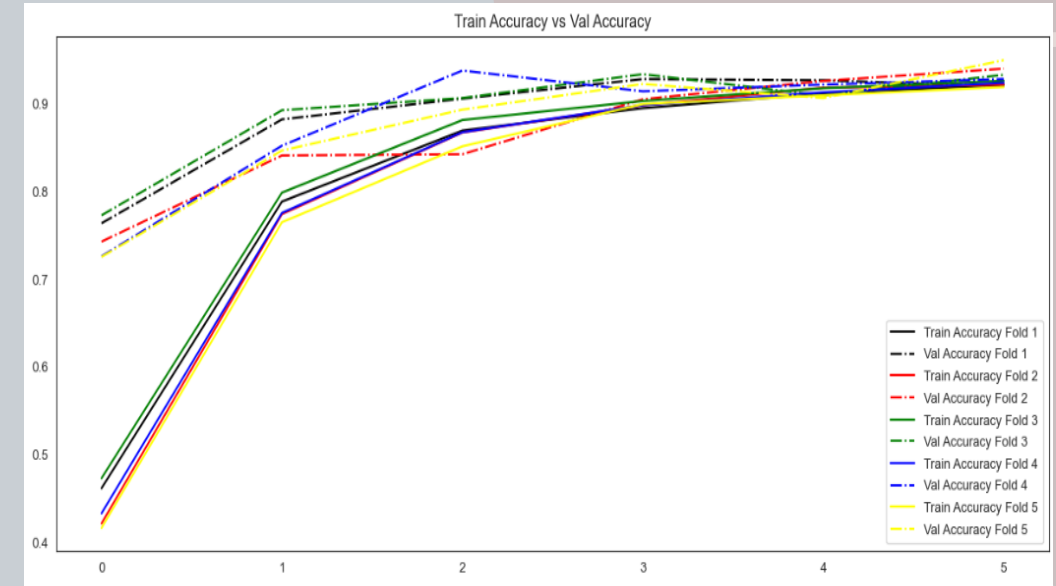
Cross-validation on Improved Baseline

Input Size 31 by 31 (Oversampled)



- Test Accuracy of 0.87
- We notice **higher validation accuracies** at least for the **earlier epochs**; however, it should converge with the train accuracies after more epochs as seen earlier
- We can say that the train and validation accuracies are quite **consistent** throughout the 5 folds, the 5 folds, which is a good sign that our model **can generalise to unseen/new data**

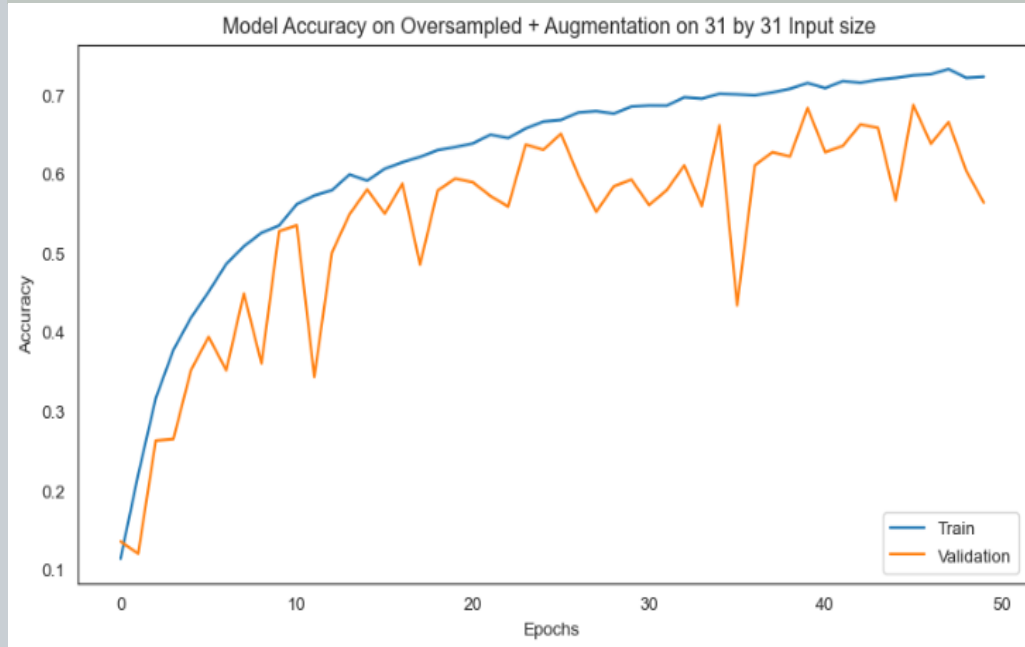
Input Size 128 by 128 (Oversampled)



- Test Accuracy of 0.9
- We do see a 'convergence' faster, at about 5 epochs for Input size of 128 by 128
- Again, train and validation accuracies are **consistent** throughout the 5 folds, which is a good sign that our model **can generalise to unseen/new data**

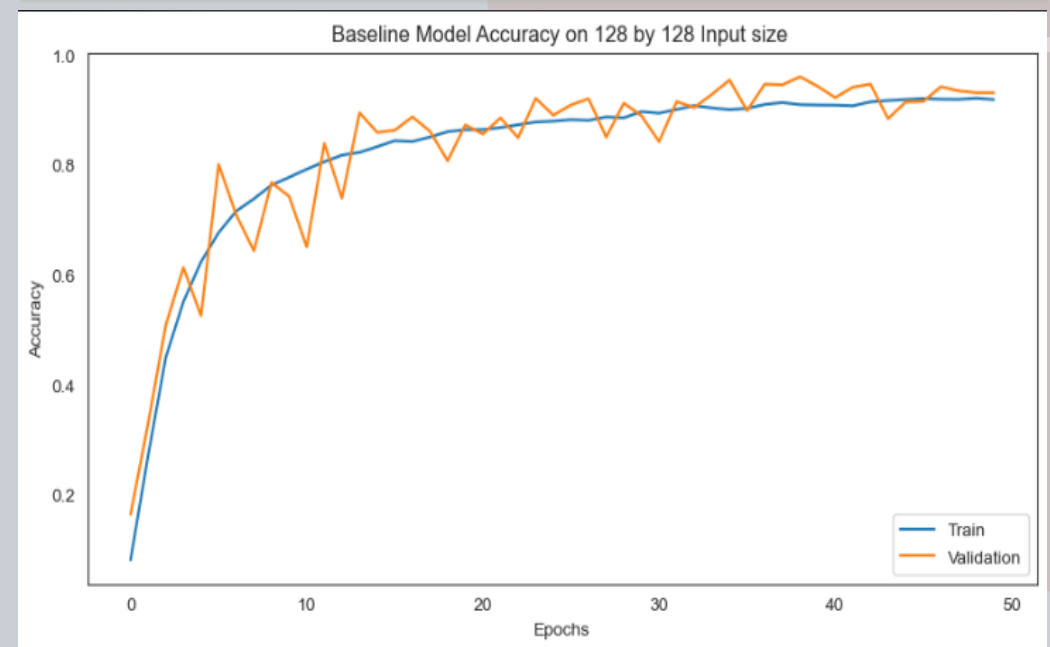
Data Augmentation Model Results

Oversampled train + Augmentation (31 Input)



- **Data Augmentation** seems to not work as well for Input size 31 by 31
- We can clearly see **overfitting**, resulting in **poor generalization** of the **validation set**
- Previously, Model seems to perform better on just **Oversampled train set**

Oversampled train + Augmentation (128 Input)



- **Data Augmentation** seems to **work way better** for 128 by 128 Input Size, **test accuracy of 0.95**
- However, we can see **fluctuations** in model's **validation** accuracy
- It possible that our Augmentation techniques may be **too harsh** / randomness introduced



Hyperparameter - Tunning

RandomSearch for 31 by 31 Inputs

Hyperparameters:

- Learning Rate
- Dropout Rate
- Activations (LeakyReLU , ReLU)

Model Architecture

- Same as what we used for our **Improved Baseline Model** (3 layers Conv2D + MaxPooling2D)
- Trained on **Oversampled** train set

Results

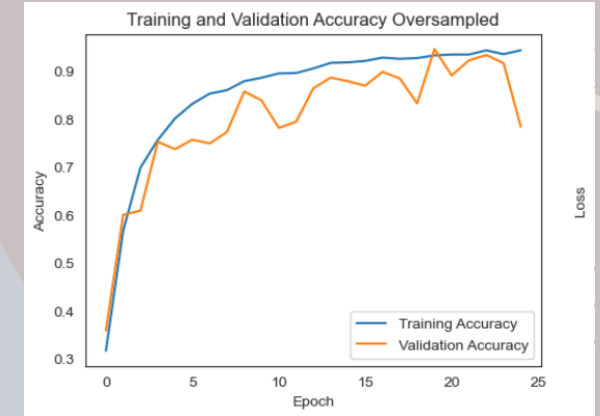
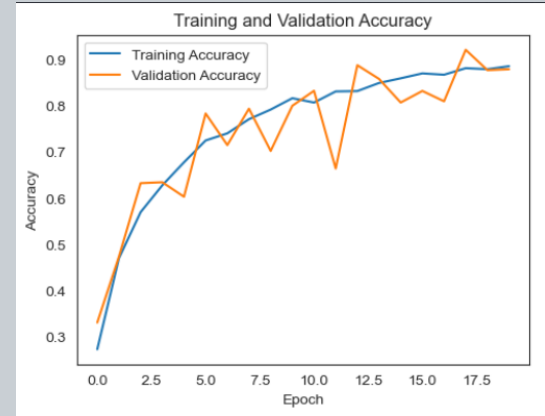
- Test Accuracy did improve by about 2% compared to Improved Baseline
- Overfitting does not really seem to be occurring
- The Validation accuracy again tend to be higher at earlier epochs but eventually converges



RandomSearch for 128 by 128 Inputs

Hyperparameters:

- Learning Rate
- Kernel regularization Rate (L1 / L2)
- Activations (LeakyReLU , ReLU)
- No. Of Layers
- Nodes



Results

- Tried RandomSearch on **Augmentation + Oversampled** as well as just on **Augmentation**
- **Augmentation + Oversampled** : 0.96 Test Accuracy
- **Augmentation**: 0.94 Test Accuracy
- We do see more **Overfitting** when we perform **oversampling** from the image on the top right
- However, **fluctuations in validation** accuracies are present for both, still indicating poor generalisation

Conclusions

Regarding Input Sizes, smaller input size takes shorter to train, has less parameters and in general, based on our 31 by 31 Input Images, was less vulnerable to overfitting as compared to 128 by 128 Images

Data Augmentation for 31 by 31 Input Images didn't seem to work as well compared to 128 by 128 Input size, could be due to non-meaningful transformations for 31 by 31 Images since they are considered very small





thank you!

Wilfred Djumin