CNN Image Classifier

Jonathan Sada

Project Overview

Objectives

- Build a Convolutional Neural Network (CNN) to classify images from a given dataset into predefined categories.
- Implement a transfer learning approach using a pre-trained model.
- Compare the performance of the custom CNN and the transfer learning model based on evaluation metrics and analysis.

Data Set

- Kaggle DataSet
- 28K medium quality animal pictures
- 10 different categories
- Images taken from Google Images

Data Set License: GPL2
Data Set Author: Corrado Alessio

Implementation Details

Analyze the data

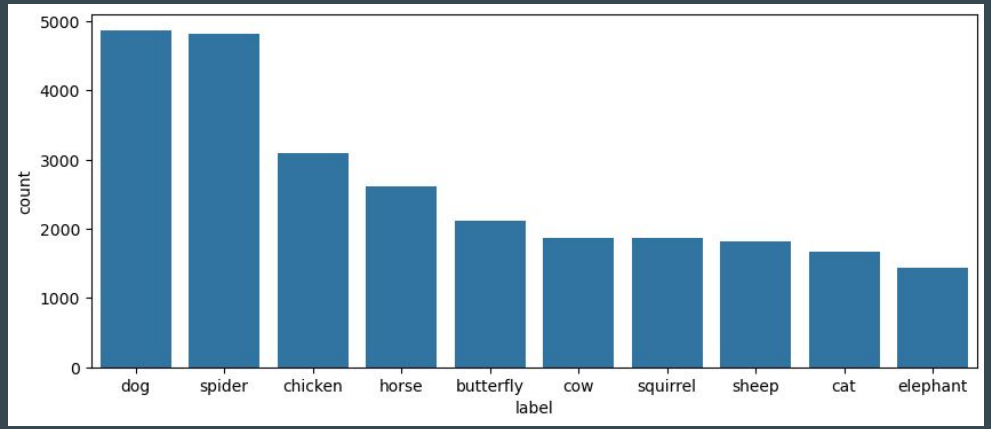
Review all the images in the folder

	img	hash	width	height	mode	format	palette	orientation	path	label
0	OIP-Gpa-3M-3IyJoYATOst0QHgHaHa.jpeg	bf7cb056645f0031e69a52aedd0d0480c286a26b	300	300	RGB	JPEG	None	squared	./dataset/raw-img/farfalla/OIP-Gpa-3M-3IyJoYAT...	butterfly
1	e832b70c2af0043ed1584d05fb1d4e9fe777ead218ac10...	3e9dad45c26c3deb96ffea72f12d2128c77050aa	640	640	RGB	JPEG	None	squared	./dataset/raw-img/farfalla/e832b70c2af0043ed15...	butterfly
2	ea36b20d2bf6083ed1584d05fb1d4e9fe777ead218ac10...	3a3ef400ebae7605cc0dcb3ebc840cad0463066b	640	480	RGB	JPEG	None	landscape	./dataset/raw-img/farfalla/ea36b20d2bf6083ed15...	butterfly
3	OIP-OYm86z-SuQPW_v8_fnzJYQHaFH.jpeg	758e7961d853872f5cfd7fb045b68f083b94f328	300	207	RGB	JPEG	None	landscape	./dataset/raw-img/farfalla/OIP-OYm86z-SuQPW_v8...	butterfly
4	OIP-sB8f5rvtA8Rvais3C20drwHaE7.jpeg	41df709171bf1bc36f1df0b8af10cbe77205d5a8	300	200	RGB	JPEG	None	landscape	./dataset/raw-img/farfalla/OIP-sB8f5rvtA8Rvais...	butterfly

26179 rows × 10 columns

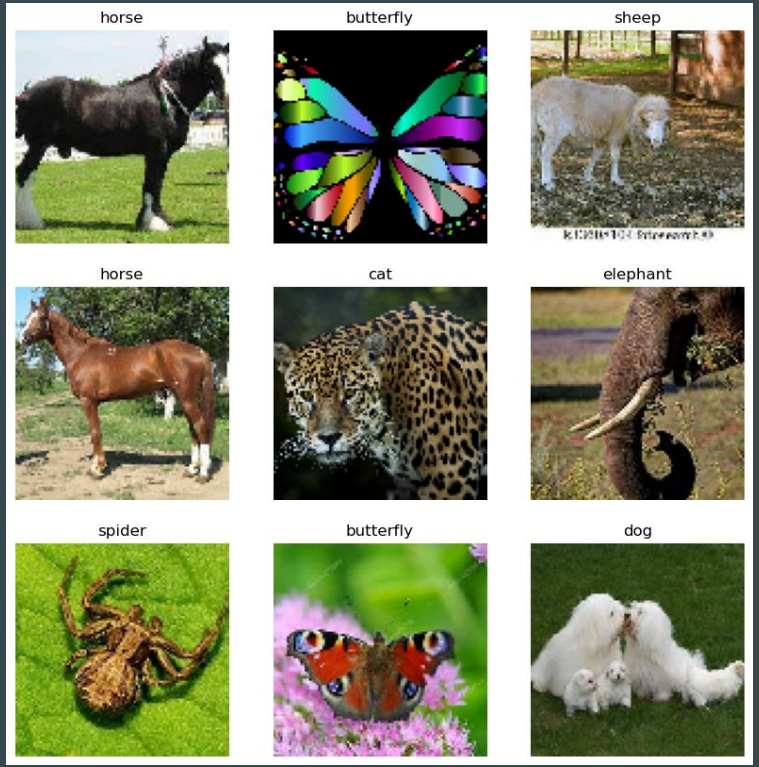
Understand the labels distribution

label	count	proportion
dog	4863	0.19
spider	4821	0.18
chicken	3098	0.12
horse	2623	0.10
butterfly	2112	0.08
cow	1866	0.07
squirrel	1862	0.07
sheep	1820	0.07
cat	1668	0.06
elephant	1446	0.06



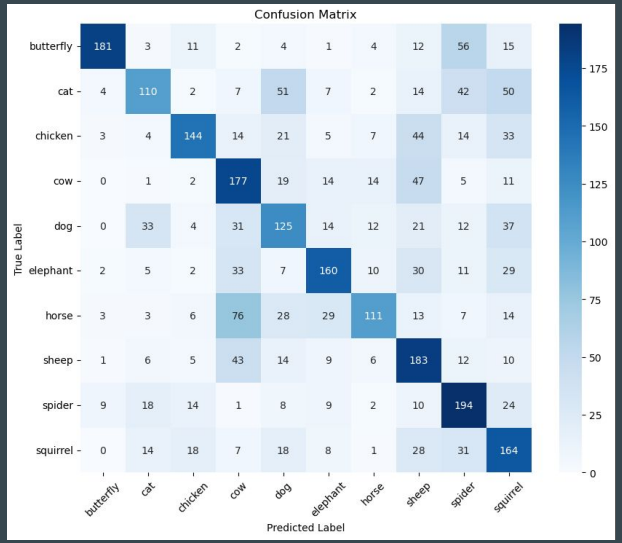
Import the data

- Balance the dataset (pandas)
  - Find the label with small items
  - Take that amount of items from each label
- Read the image (keras.load_img)
  - Setted a default size of 100 x 100
  - Resize and Crop logic
- Split the dataset (sklearn.train_test_split)
  - Split the data on a 80-20 proportion



Model 1: Base Model

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 98, 98, 32)	896
max_pooling2d (MaxPooling2D)	(None, 49, 49, 32)	0
conv2d_1 (Conv2D)	(None, 47, 47, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 23, 23, 64)	0
flatten (Flatten)	(None, 33856)	0
dense (Dense)	(None, 128)	4,333,696
dense_1 (Dense)	(None, 10)	1,290



Details:

- 2 Convolutional layers.
- Epoch 60, Batch size 24.
- Early stop.
- Data Balanced.

Conclusions:

- Model used as a base for comparison
- Iteratively tested to evaluate:
  - Resize vs resize and crop
  - Image size
  - Sorting data before balance

accuracy_train	loss_train	val_accuracy_train	val_loss_train
0.881051	0.383454	0.516943	1.922947

accuracy_test	precision_test	recall_test	f1-score_test
0.547026	0.577813	0.547026	0.547615

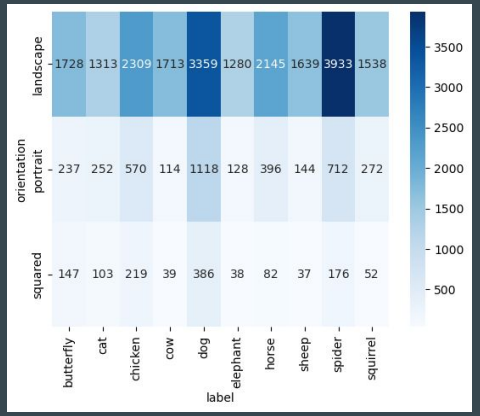
Exploratory Data Analysis

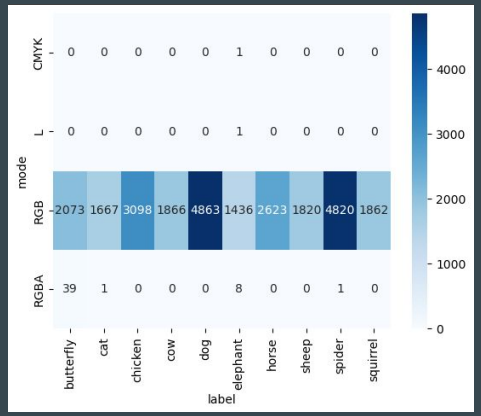
Overall Review:

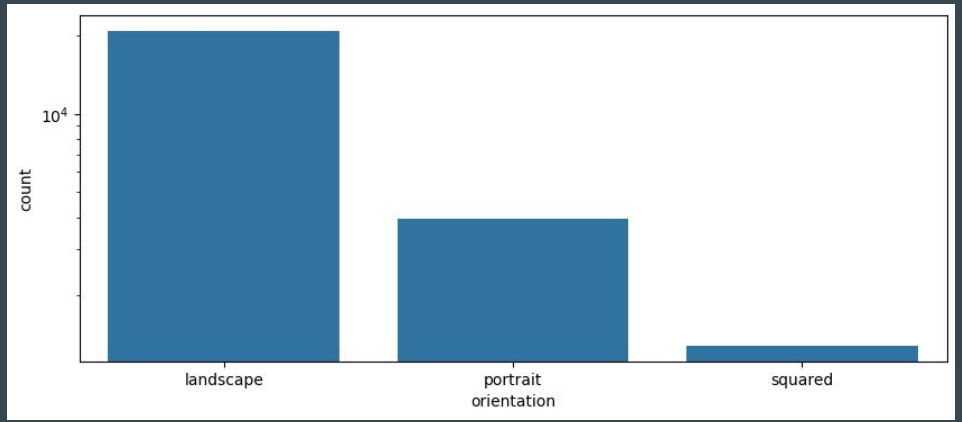
- No duplicated images (based on hashes)
- Palette property was not useful (empty)

Univariate and Bivariate Analysis:

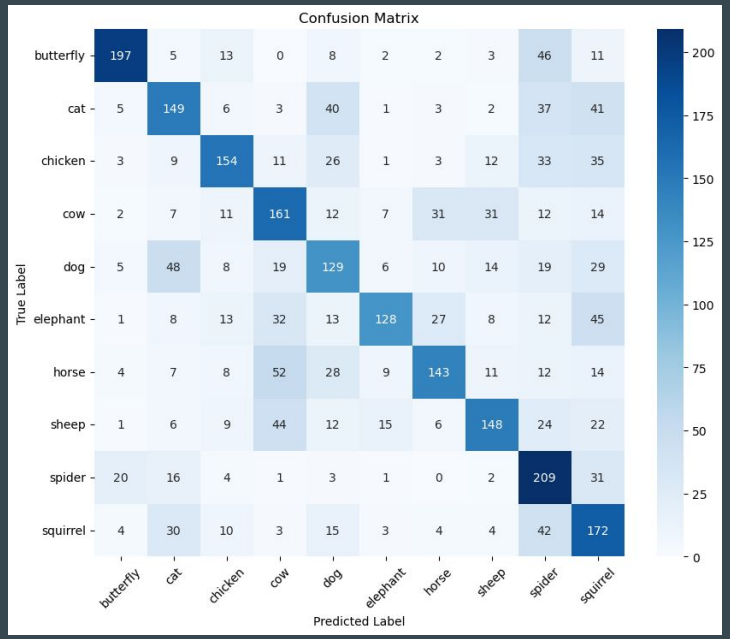
- Majority of images where RGB and JPEG (all except 51)
- 80% of images are landscape
- 300 is the most common width and height value but:
  - 300x225 the most common resolution (17%)
  - 45% of the resolutions have less than 1% of images.
  - 16 resolutions have 1% or more images.







Model 2: Improve Dataset



Details:

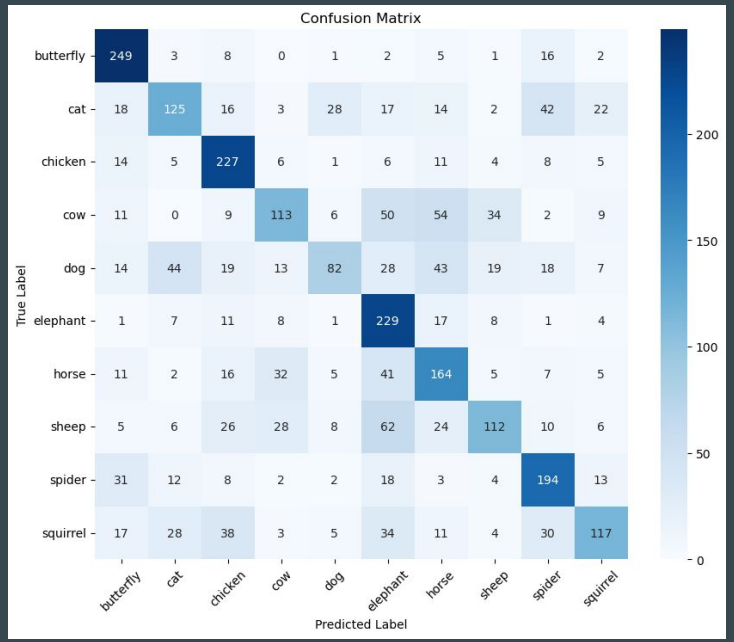
- Same architecture than model 1.
- Changed dataset (iteratively):
  - Keep only landscapes resolutions.
  - Keep only JPG and RGB images.

Conclusions:

- Not relevant improvement by changing these features.
- Slightly improvement removing non JPEG images and non RGB

	model	accuracy_train	loss_train	val_accuracy_train	val_loss_train	accuracy_test	precision_test	recall_test	f1-score_test
0	model_01	0.881051	0.383454	0.516943	1.922947	0.547026	0.577813	0.547026	0.547615
1	model_02	0.889015	0.355193	0.540042	1.980298	0.553621	0.581235	0.553621	0.556701

Model 3: Data Augmentation



Details:

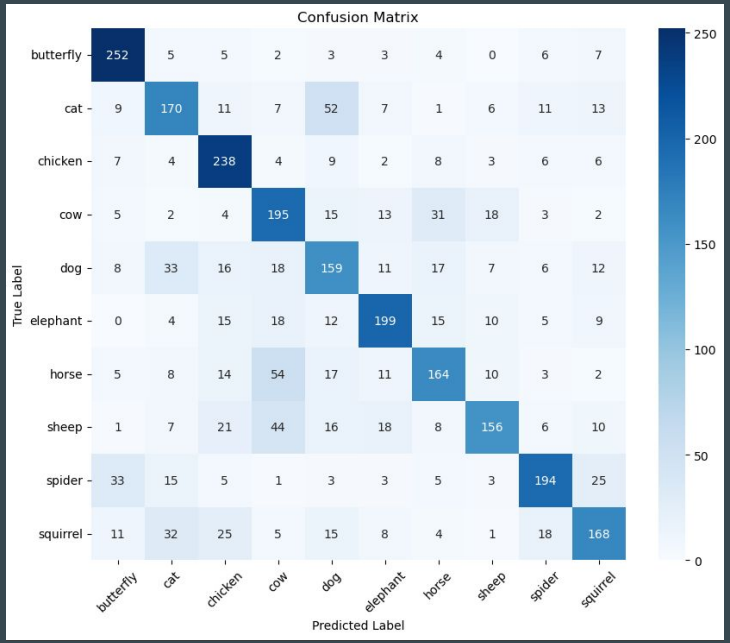
- Same architecture than model 1 and 2.
- Using resulting dataset from Model 2:
  - Kept all the data except non JPEG and non RGB
- Implemented data augmentation

Conclusions:

- Data augmentation improved slightly the accuracy.
- The overfitting show a high decrease.

	model	accuracy_train	loss_train	val_accuracy_train	val_loss_train	accuracy_test	precision_test	recall_test	f1-score_test
1	model_02	0.889015	0.355193	0.540042	1.980298	0.553621	0.581235	0.553621	0.556701
2	model_03	0.501480	1.415400	0.547006	1.322050	0.561281	0.567505	0.561281	0.545866

Model 4: Model Tweaking



Details:

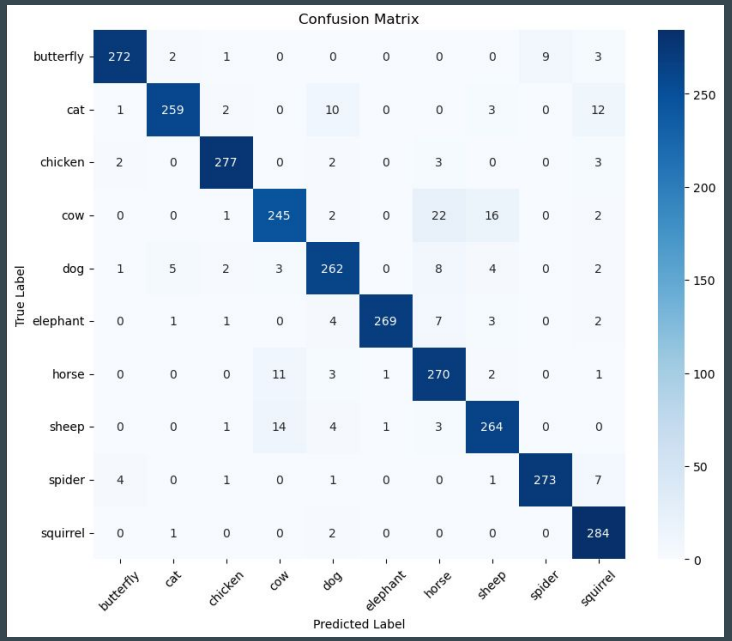
- Architecture was incremented with 2 convolutional layers (128 and 256)
- Some parameters of the model where changed:
  - learning_rate: 0.001
  - epoch: 60
  - batch_size: 128

Conclusions:

- There is a relevant improvement in accuracy
- Overfitting decreased slightly

	model	accuracy_train	loss_train	val_accuracy_train	val_loss_train	accuracy_test	precision_test	recall_test	f1-score_test
2	model_03	0.501480	1.415400	0.547006	1.322050	0.561281	0.567505	0.561281	0.545866
3	model_04	0.608635	1.119599	0.628134	1.208655	0.659819	0.663302	0.659819	0.657722

Model 5: Transfer Learning (MobileNetV2)



Details:

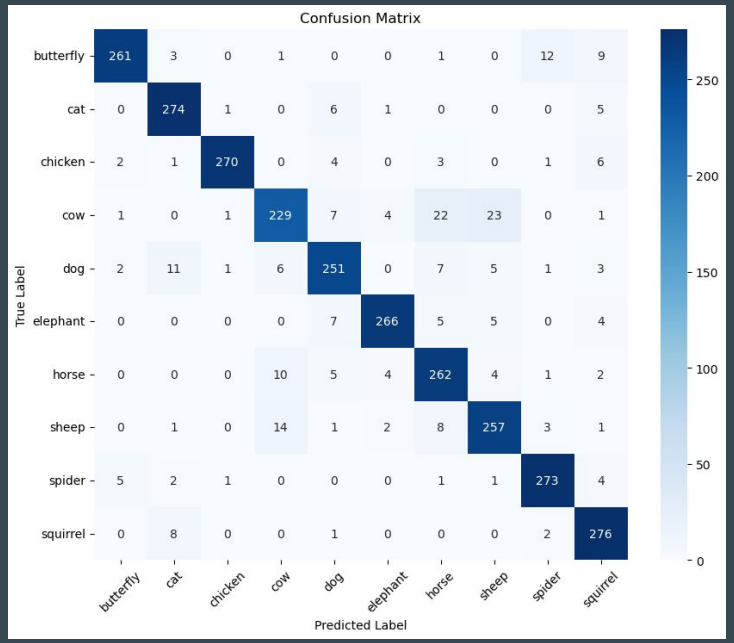
- Created a model with a MobileNetV2 instance and 3 more layers:
  - Global Average Pooling 2D
  - Dense 128 Neurons
  - Dense 10 Neurons (Output layer)
- Not using data augmentation
- Same hyperparameters

Conclusions:

- The performance of the model increased dramatically
- Overfitting increased slightly (acceptable)
- The fit time was reduced from minutes to seconds

	model	accuracy_train	loss_train	val_accuracy_train	val_loss_train	accuracy_test	precision_test	recall_test	f1-score_test
3	model_04	0.608635	1.119599	0.628134	1.208655	0.659819	0.663302	0.659819	0.657722
4	mnv2_model	0.990947	0.024827	0.919916	0.460934	0.931407	0.932998	0.931407	0.931542

Model 6: Transfer Learning (EfficientNetV2B3)



Details:

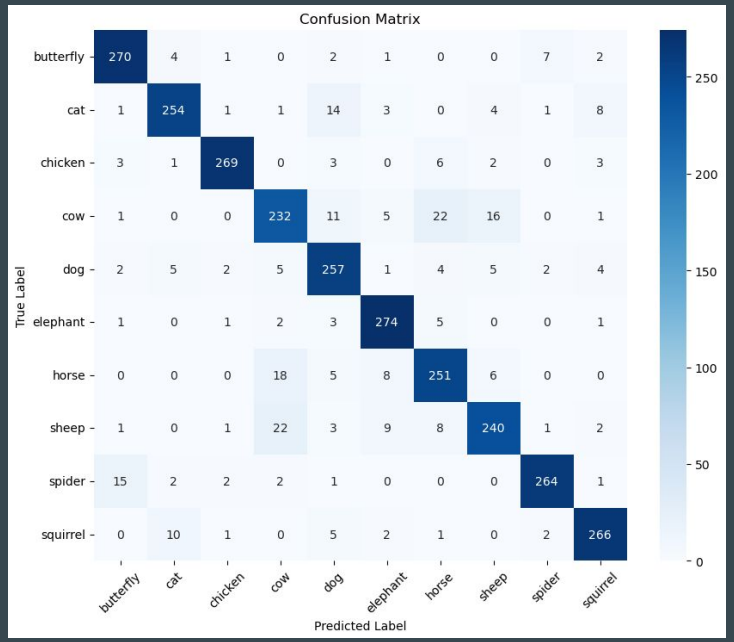
- With a similar architecture than MobileNetV2
- Using custom preprocessing
- Adjusting hyperparameters:
  - epochs: 10
  - batch_size: 32

Conclusions:

- Not as good results as MobileNet but still very relevant

	model	accuracy_train	loss_train	val_accuracy_train	val_loss_train	accuracy_test	precision_test	recall_test	f1-score_test
4	mnv2_model	0.990947	0.024827	0.919916	0.460934	0.931407	0.932998	0.931407	0.931542
5	enb3_model	0.999826	0.004057	0.912953	0.380537	0.911908	0.913091	0.911908	0.911785

Model 7: Transfer Learning (DenseNet121)



Details:

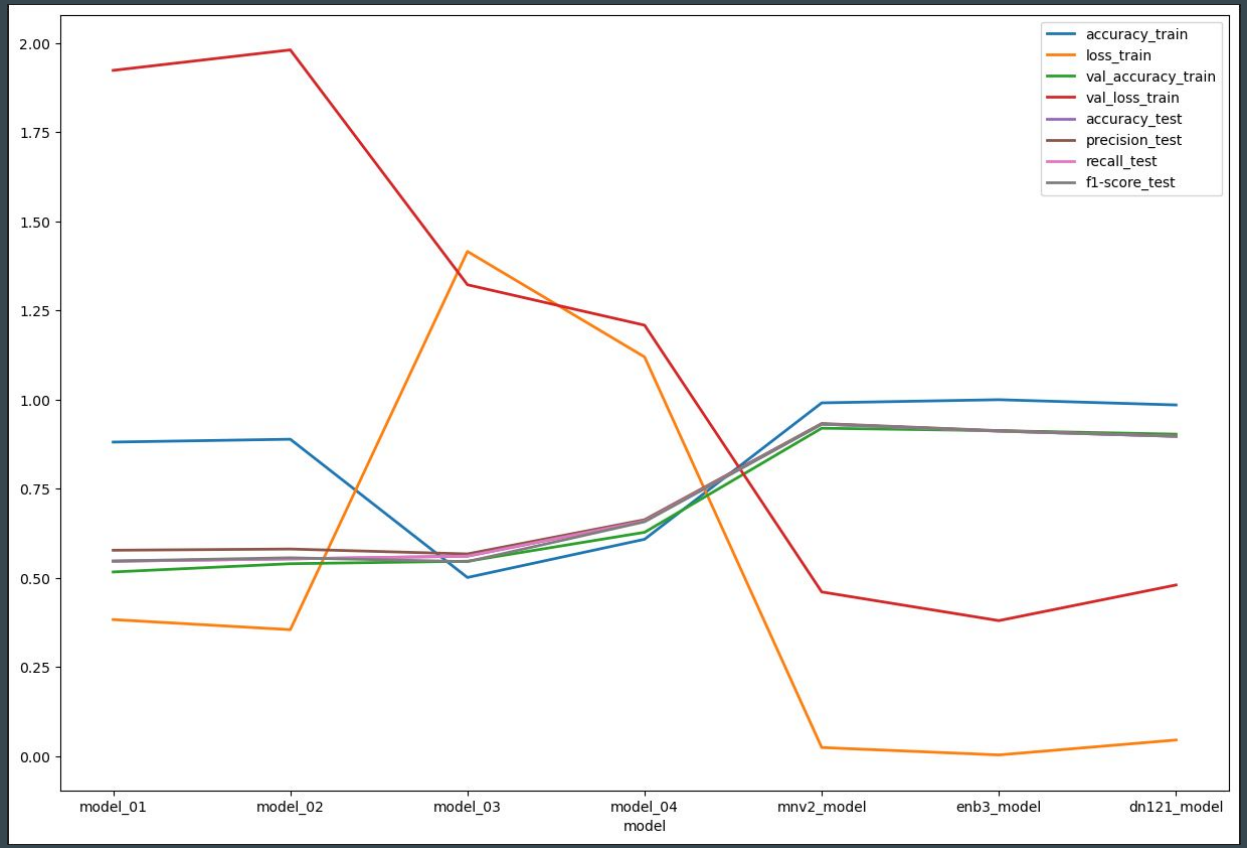
- With a similar architecture than MobileNetV2
- Same hyperparameters EfficientNetV2B3

Conclusions:

- Worst of results of the 3 models tested with transfer learning
- Still good results considering the dataset

	model	accuracy_train	loss_train	val_accuracy_train	val_loss_train	accuracy_test	precision_test	recall_test	f1-score_test
5	enb3_model	0.999826	0.004057	0.912953	0.380537	0.911908	0.913091	0.911908	0.911785
6	dn121_model	0.985115	0.045904	0.903203	0.480451	0.897284	0.897914	0.897284	0.897274

Model Comparison



Major Obstacle

Major Obstacle

- Memory management
- Execution time for some processes (data related, fitting, ...).
- Analysis over non-deterministic models (comparison)
- Understand what's failing (preprocess_input) in between many parameters

Conclusion and Insights

Conclusion and Insights

- Make models deterministic during training and non-deterministic on deploy.
- Iterative improvement, one think every time.
- Take notes of the tests and results.
- Understand well what are the needs of the pre-train model.

Late improvements.

Late Improvements

After finishing this presentation I worked on improve some parts I was unhappy with:

- Image Cropping.
- Label Balancer.
- Make models deterministic.
- Increase transfer learning models depth.
- Fine-Tuning.
- Evaluate EDA data filtering for further models.

Late Improvements

Previous results:

	model	accuracy_train	loss_train	val_accuracy_train	val_loss_train	accuracy_test	precision_test	recall_test	f1-score_test
0	model_01	0.881051	0.383454	0.516943	1.922947	0.547026	0.577813	0.547026	0.547615
1	model_02	0.889015	0.355193	0.540042	1.980298	0.553621	0.581235	0.553621	0.556701
2	model_03	0.501480	1.415400	0.547006	1.322050	0.561281	0.567505	0.561281	0.545866
3	model_04	0.608635	1.119599	0.628134	1.208655	0.659819	0.663302	0.659819	0.657722
4	mnv2_model	0.990947	0.024827	0.919916	0.460934	0.931407	0.932998	0.931407	0.931542
5	enb3_model	0.999826	0.004057	0.912953	0.380537	0.911908	0.913091	0.911908	0.911785
6	dn121_model	0.985115	0.045904	0.903203	0.480451	0.897284	0.897914	0.897284	0.897274

Result after changes:

	model	accuracy_train	loss_train	val_accuracy_train	val_loss_train	accuracy_test	precision_test	recall_test	f1-score_test
0	model_01	0.830999	0.521517	0.501729	1.815209	0.529046	0.560268	0.529046	0.532693
1	model_02	0.806146	0.611757	0.498607	1.881110	0.521936	0.542052	0.521936	0.520516
2	model_03	0.501480	1.431633	0.544916	1.328481	0.564415	0.560131	0.564415	0.553063
3	model_04	0.642671	1.019825	0.701253	0.995752	0.702646	0.712956	0.702646	0.699232
4	mnv2_model	0.989815	0.032202	0.926880	0.416603	0.931058	0.932087	0.931058	0.931222
5	mnv2_model (fine tune)	0.971382	0.094761	0.970409	0.106102	0.936630	0.936988	0.936630	0.936687
6	enb3_model	1.000000	0.002064	0.916086	0.369216	0.915042	0.916372	0.915042	0.915362
7	dn121_model	0.982852	0.049107	0.903552	0.482274	0.908078	0.910481	0.908078	0.908263

Thank you.