Mincu Adrian-Lucian
Grupa 232

Realistic Image Classification

Mincu Adrian-Lucian
Grupa 232

Table of contents

Mincu Adrian-Lucian
Grupa 232

Challenges Regarding Data

Black and White Images

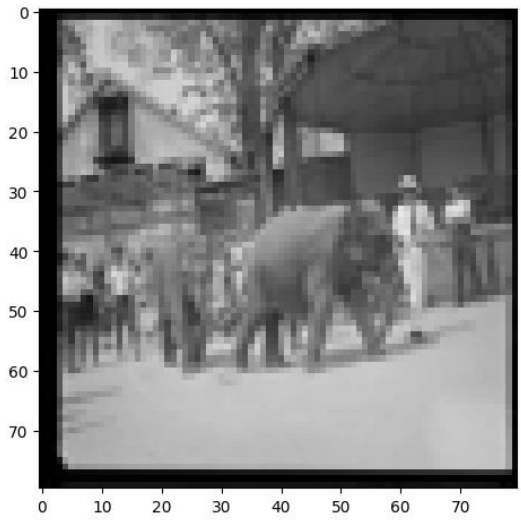
During my data loading process I have observed that some of my vectors didn't have the same shape. The reason? Black and white images only had one color channel instead of 3 (RGB).

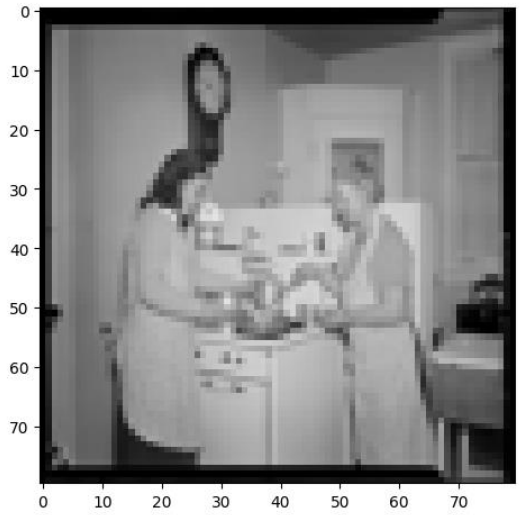
As for the solution, I identified all black and white images and for each and every one I duplicated the singleton value 3 times to replicate the RGB color model.

Black Margin

Some images, the ones I found were only black and white, had a black border which was inconsistent in size. I thought that maybe eliminating the border that didn't contain any information would improve my model.

Example images:





I have experimented with different crop amounts to see which one will be best. The table below showcases my models performance using different number of pixels to exclude from the image:

Resized Image Shape	Accuracy	Loss
70 x 70 px	0.7223	0.7311
74 x 74 px	0.7256	0.7024
80 x 80 px	0.7340	0.7222

We can see that cropping the images resulted in a worse performance overall, so I decided to keep the original images.

Feature Selection

For this challenge I went with the pixel values of the image. I thought that given the fact that this model should predict realistic images then being able to see patterns and color values will benefit more than a color histogram for example. I have not tried any other method.

Mincu Adrian-Lucian
Grupa 232

Normalization

For the normalization of the data I only tried one method and that is dividing each pixel value by 255 to get a range between 0 and 1. After discussions with my lab professor and some online research I came to the conclusion that the normalization method will not have a powerful outcome in my models performance.

Other Preprocessing

I've also constructed a larger dataset containing the train images alongside validation images to be used when training for the final prediction on the test dataset. And another method of precaution was to shuffle all the data to make sure there will an unbiased training.

Loss Function

Given the nature of this task (multiclass) I was left to choose between a CategoricalCrossentropy loss and a SparseCategoricalCrossentropy loss. What's the difference?

Well the first one is used when labels are provided as one-hot representation, meaning that the labels come in a binary vector which has the dimension of the number of classes and if v[i] = 1 then it means that the image has label i. On the other hand in sparse categorical crossentropy loss the labels come as integers.

Models

SVM

Data Loading

I went trough every image and loaded it using the skimage API. While doing this I also resized them to a 1 x 1 px size, because as I have learnt later it would take a lot of time to train a SVM model on 80 x 80 px images.

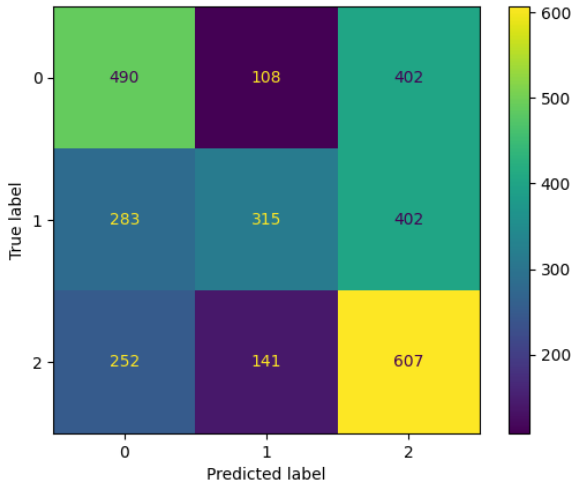
Data Preparation

Because this is a simpler model and doesn't flatten your images automatically and because sklearn algorithms expect a matrix for the input I had to manualy flatten the datasets.

Architecture

The model itself was used only to complete the minimum requirements of having at least 2 models, and so I went with no parameters and no tuning on it, just to get it over the baseline.

Confusion Matrix



Mincu Adrian-Lucian
Grupa 232

CNN

Starting Architecture

Layer	Parameters
Conv2D	filters=32, kernel_size=(3,3)
MaxPooling2D	pool_size=(2,2)
Conv2D	filters=64, kernel_size=(3,3)
MaxPooling2D	pool_sixe=(2,2)
Conv2D	filters=128, kernel_size=(3,3)
MaxPooling2D	pool_size=(2,2)
Flatten	-
Dense	units=128
Dense	units=3, activation=softmax

Activation Functions Trials

Troughout all trials I've used 10 epochs and the starting architecture presented above.

Activation Function	Accuracy	Loss
All relu	0.5630	1.1789
All gelu	0.5196	2.1936
All tahn	0.4343	2.4040
All sigmoid	0.3333	1.0992
All leaky_relu	0.5529	1.2542
All relu6	0.5693	1.1533
All elu	0.4566	2.3053
relu6 -> relu6 -> relu6 -> relu	0.5889	0.9824
relu6 -> relu6 -> relu -> relu	0.5896	1.0468
relu6 -> relu -> relu6 -> relu	0.5973	0.9587
relu -> relu6 -> relu -> relu6	0.5873	0.9970
gelu -> relu6 -> gelu -> relu6	0.5406	1.3167
leaky_relu -> relu6 -> leaky_relu -> relu6	0.5429	1.5441
relu6 -> gelu -> relu6 -> gelu	0.5613	1.3527
relu6 -> leaky_relu -> relu6 -> leaky_relu	0.5780	1.0550

From this table there are several conclusion to be drawn. First the sigmoid function was the worst, given that the dataset is balanced it had an accuracy of 0.33 and having 3 classes means it woud just pick a random class for each image. The best ones were relu and relu6 and I decided to go further with them and different combinations of those 2 functions.

Mincu Adrian-Lucian
Grupa 232

Dropout Trials, relu6, relu

I have went with the current activations: relu6 -> relu6 -> relu -> relu -> softmax

I have only introduced a dropout layer after the Flatten and before the first Dense layer and it's values and obtained accuracy/loss can be seen in the table below.

Dropout	Accuracy	Loss
0.5	0.6370	0.8254
0.25	0.6063	0.8927
0.75	0.6276	0.8257
0.625	0.6176	0.8485

During these experiments I have decided to go further with the 0.5 dropout layer as it provided the best results so far and tried to experiment with BatchNormalization layers with this given architecture. Also, a thing to keep in mind is that because the model performance got better with a Dropout layer, then it means that my previous model was overfitting.

BatchNorm Placement	Epochs	Accuracy	Loss
After activation functions on Conv2D	10	0.6066	1.0274
After activation funcitons on Conv2D and Dense	10	0.5690	1.0947
After activation functions on Dense	10	0.6156	0.8430
After activation functions on Dense	20	0.6620	0.7695

Here I have concluded that 10 epochs were way to low, and in some cases it made my model either be really good or really bad. Why? Well because the learning rate was quite high, something I will find out later and the validation accuracy was really spiky which made it very unstable in generating a result.

I have submitted this current model and managed to get a 0.6044 on the test dataset.

Mincu Adrian-Lucian
Grupa 232

Overfitting Problem

We have seen before that my model was overfitting and thus couldn't reach an accuracy of more than 0.6044 on the test dataset.

This is why I decided to change things up a little bit and reduce the number of neurons and add more layers of dropout. The current architecture can be seen here and it was generated by ChatGPT so I would not waste time writing word tables by hand.

Layer Type	Output Shape	Parameters	Activation Function	Additional Information
Conv2D	(78, 78, 32)	896	ReLU6	kernel_size=(3,3), input_shape=(80, 80, 3)
BatchNormalization	(78, 78, 32)	128	-	-
MaxPooling2D	(39, 39, 32)	0	-	-
Conv2D	(37, 37, 64)	18,496	ReLU6	kernel_size=(3,3)
BatchNormalization	(37, 37, 64)	256	-	-
MaxPooling2D	(18, 18, 64)	0	-	-
Flatten	(20,736)	0	-	-
Dropout	(20,736)	0	-	rate=0.3
Dense	(64)	1,327,168	ReLU	-
BatchNormalization	(64)	256	-	-
Dropout	(64)	0	-	rate=0.5
Dense	(3)	195	Softmax	-

With this configuration I managed to get an accuracy of 0.6333 and loss of 0.9562. It doesn't seem like much changed but I am going to experiment more with the 2 Dropout layers.

Dropout	Accuracy	Loss
0.3 -> 0.5	0.6333	0.9562
0.2 -> 0.7	0.5696	1.1490

Increasing the dropout didn't seem to go that well. I figured if decreasing the layers neutrons and increasing dropout didn't help me then I will try other methods of combatting overfitting. So I added L1 regularization and decreases the learning rate. This seemed to work as I reached an accuracy of 0.6743 and loss of 0.9987.

I've changed number of epochs to be 100 to get a more stable result because in the first 50 epochs the model is really unstable, but then it stabilizes and plateuas.

L1	Learning rate	Accuracy	Loss
0.001	0.00001	0.5696	1.1499
Default	0.0001	0.6743	0.9987
0.005	0.0005	0.6970	1.0828
0.005	0.00025	0.6596	0.9575
0.0075	0.00025	0.6690	0.8931
0.02	0.00025	0.7099	0.7368

From this table we can see that a higher regularization works best for my model. I have used only L1 as I read on this website that L1 provides a more sparse solution meaning that many coefficient can become 0, resulting a a feature selection method, whereas L2 evenly distributes the values.

With the given L1 and learning rate I submitted the model and got a 0.68 on the test dataset.

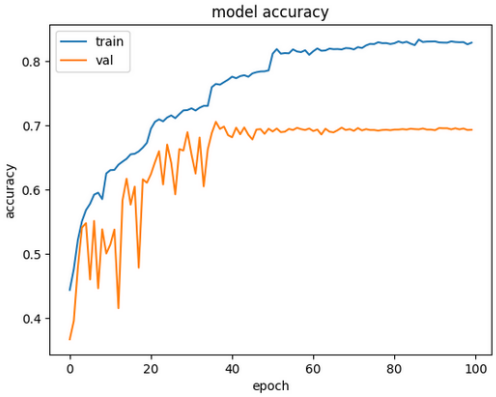
Dropout Tuning

I started tuning the dropout layers, the two in this case which we can see from our previous model using a simple for in for:

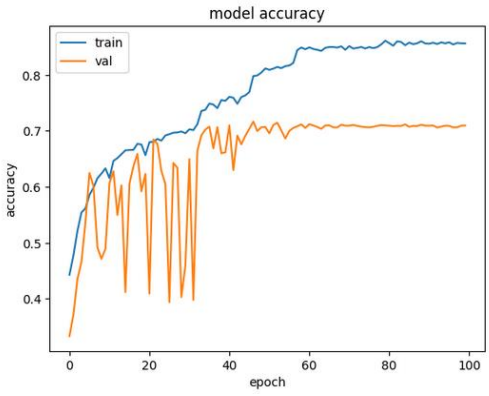
```
for d1 in [0.2, 0.4, 0.6, 0.8]:
    for d2 in [0.2, 0.4, 0.6, 0.8]:
```

And here are the result:

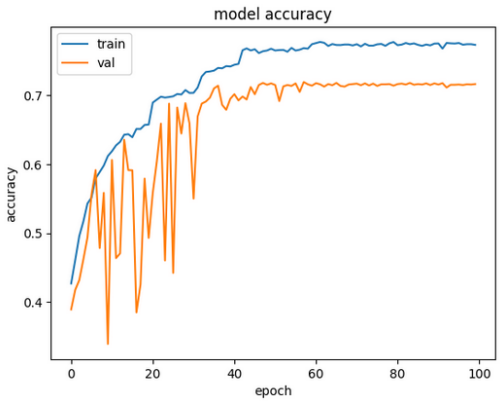
```
DROPOUT1: 0.2
DROPOUT2: 0.2
```



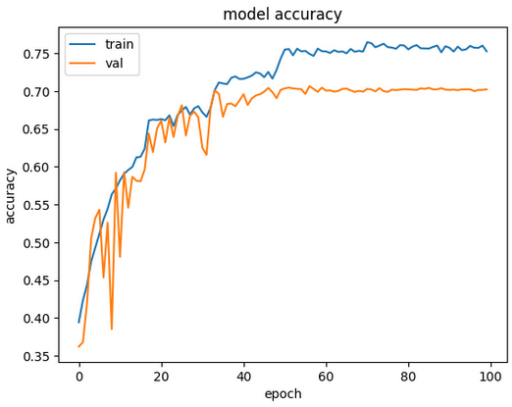

```
DROPOUT1: 0.2
DROPOUT2: 0.4
```




```
DROPOUT1: 0.2
DROPOUT2: 0.6
```



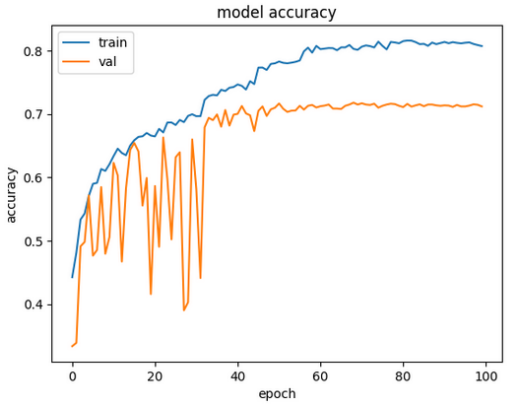

```
DROPOUT1: 0.2
DROPOUT2: 0.8
```

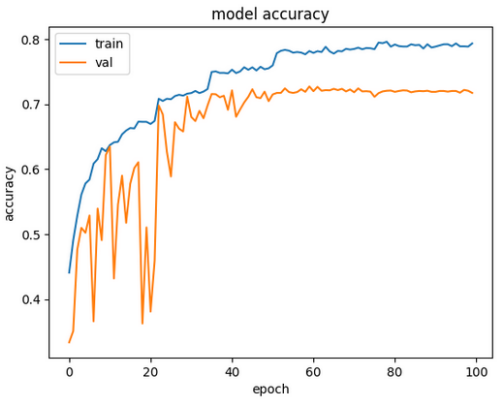



```
DROPOUT1: 0.4
DROPOUT2: 0.2
```

```
DROPOUT1: 0.4
DROPOUT2: 0.4
```

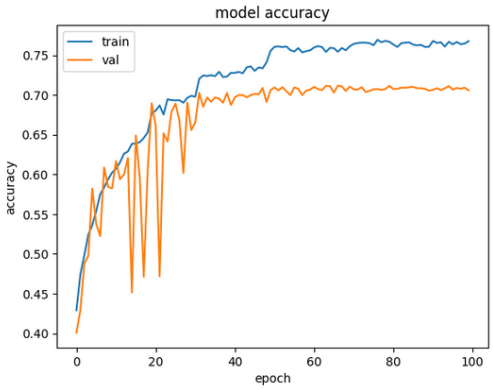
Mincu Adrian-Lucian
Grupa 232

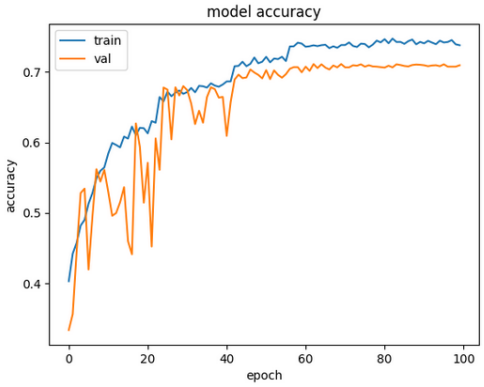





```
DROPOUT1: 0.4
DROPOUT2: 0.6
```

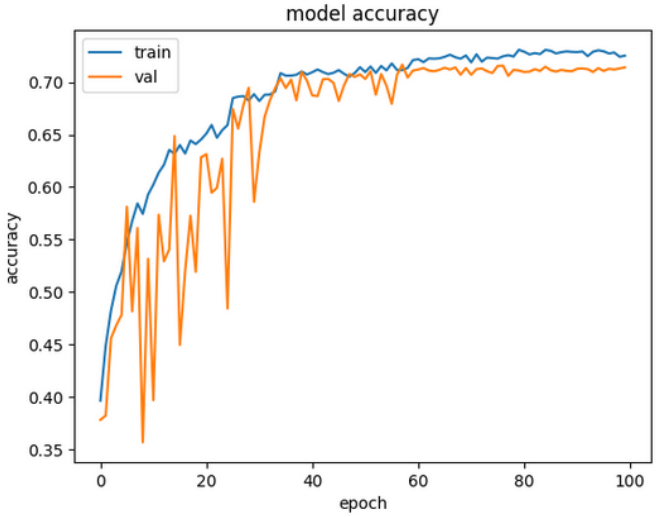
```
DROPOUT1: 0.4
DROPOUT2: 0.8
```





And so on for the others. I don't think it's worth to bore you with all these plots. The ideea is that the best one was this:

```
DROPOUT1: 0.8
DROPOUT2: 0.2
```



Underfitting problem

And in this case, it seemed like we fixed overfitting, but there is another problem, now it looks like it's underfitting because the training accuracy tops up at about 0.72. I increased the neurons in each layer and with this configuration:

Layer (type)	Output Shape	Param #	Activation	Regularizer
Conv2D (Conv2D)	(None, 78, 78, 64)	1792	ReLU6	None
BatchNormalization (BatchNormalization)	(None, 78, 78, 64)	256	None	None
MaxPooling2D (MaxPooling2D)	(None, 39, 39, 64)	0	None	None
Conv2D (Conv2D)	(None, 37, 37, 128)	73856	ReLU6	None
BatchNormalization (BatchNormalization)	(None, 37, 37, 128)	512	None	None
MaxPooling2D (MaxPooling2D)	(None, 18, 18, 128)	0	None	None
Conv2D (Conv2D)	(None, 16, 16, 256)	295168	ReLU6	None
BatchNormalization (BatchNormalization)	(None, 16, 16, 256)	1024	None	None
MaxPooling2D (MaxPooling2D)	(None, 8, 8, 256)	0	None	None
Flatten (Flatten)	(None, 16384)	0	None	None
Dropout (Dropout)	(None, 16384)	0	None	None
Dense (Dense)	(None, 256)	4194560	ReLU	L1 (0.005)
BatchNormalization (BatchNormalization)	(None, 256)	1024	None	None
Dropout (Dropout)	(None, 256)	0	None	None
Dense (Dense)	(None, 3)	771	Softmax	None

And a learning rate of 0.0001 plus a callback of learning rate scheduler with a patience of 5, factor of 0.2 and min_lr=1e-7 I managed to pull off a 0.75 of the test dataset.

Next up I tried tuning the batch size, a lower batch size can help the model converge faster, but train longer, while a larger one make it converge slower, but train faster.

Batch Size	Accuracy	Loss
64	0.7223	0.8237
32	0.7416	0.7161
256	0.6816	1.4765
512	0.7066	1.1001
16	0.7419	0.6705

And in this case, it seems like a lower batch size also helped my model, increasing accuracy and reducing the loss. I chose to go with the batch size of 16 over the 32 because it had a slightly lower loss value.

Mincu Adrian-Lucian
Grupa 232

Final Architecture

The last thing I experimented with was using padding on the Conv2D layers, which would help maintain the dimensions of the images troughout each filter, making sure I don't lose potential data

I have also doubled the neurons in the second Conv2D layer and with this configuration:

Layer Type	Output Shape	Parameters	Activation Function	Additional Details
Input	(80, 80, 3)	0	-	
Conv2D (128 filters)	(80, 80, 128)	3,584	ReLU6	kernel_size=(3,3), padding='same', input_shape=(80, 80, 3)
BatchNormalization	(80, 80, 128)	512	-	
MaxPooling2D	(40, 40, 128)	0	-	pool_size=(2,2)
Conv2D (256 filters)	(40, 40, 256)	295,168	ReLU6	kernel_size=(3,3), padding='same'
BatchNormalization	(40, 40, 256)	1,024	-	
MaxPooling2D	(20, 20, 256)	0	-	pool_size=(2,2)
Conv2D (256 filters)	(20, 20, 256)	590,080	ReLU6	kernel_size=(3,3), padding='same'
BatchNormalization	(20, 20, 256)	1,024	-	
MaxPooling2D	(10, 10, 256)	0	-	pool_size=(2,2)
Flatten	(25,600)	0	-	
Dropout	(25,600)	0	-	rate=0.8
Dense (256 units)	(256)	6,553,856	ReLU	kernel_regularizer=L1(0.005)
BatchNormalization	(256)	1,024	-	
Dropout	(256)	0	-	rate=0.2
Dense (3 units)	(3)	771	Softmax	

I've managed to get 0.75 on the final private dataset.

Confusion Matrix