



Faculty of Engineering
Credit Hour System



Cairo University



CNN CLASSIFICATION REPORT

Pattern Recognition final project



JUNE 12, 2021

Team members:

Abdulrahman Hussam Fathi – 1170371

Hisham Khaled Al Gendy- 1170479

Rawda Ruameih- 1170448

Contents

1. Problem definition and importance.....	2
1.1 Brief about CNN	2
1.2 Caltech101	2
2. Methods.....	2
2.1 Layers Brief:.....	2
2.2 Techniques Brief:	3
2.2.1 Activation Function:	3
2.2.2 Dropout	3
2.3 Calculations	3
3. Algorithms.....	4
3.1 Keras Model	4
Google Collab Specs & time elapsed.....	6
Visualizing the feature maps:.....	7
3.2 Alex-Net:	8
First convolution layer: C1	8
First Max Pooling Layer: S2	8
Second Convolution Layer: C3	8
Second Max Pooling Layer: S4	9
Third Convolution Layer: C5.....	9
Fourth Convolution Layer: C6	9
Fifth Convolution Layer: C7.....	10
Third Max Pooling Layer: S8.....	10
First Fully Connected Layer: F9	10
Second Fully Connected Layer: F10	11
Third Fully Connected Layer: F11.....	11
4. Results and Discussion	12
4.1 Keras Model	12
4.2 Alex-Net	14
4.3 Some examples for incorrect classification:	16
4.4 Some examples for correct classification:	17
Appendix	18
Codes used.....	18

1. Problem definition and importance

Our aim in this project is to design a CNN (Convolutional Neural Network) based classifier that classified between 10 image categories using the concepts grasped in the course and shows how we could select the optimal structure for our network. In addition, to using an open source classifier and comparing between the two approaches using **Caltech 101** data set.

1.1 Brief about CNN

CNN is a Deep Learning algorithm that takes in an input image, assign learnable weights and biases to various objects in the image to be able to differentiate one from the other. The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction.

1.2 Caltech101

The Caltech 101 data set consists of a total 9,146 images which split between 101 different object categories. Each object category contains between 40 and 800 images where every image is about 300x200 pixels.

In our project we used:

['Faces' 'Leopards' 'Motorbikes' 'airplanes' 'bonsai' 'car' 'chandelier' 'ketch' 'revolver' 'watch']

2. Methods

2.1 Layers Brief:

- **Convolutional layer:** convolution operation occurs between the filters and the image passed through a convolutional neural network
- **Batch Normalization:** mitigates the effect of unstable gradients through standardizing and normalizing the input values.
- **Max Pooling:** maximum pixel value of pixels that fall within the receptive field of a unit within a sub-sampling layer is taken as the output.
- **Flatten Array:** flattens the input image data into a one-dimensional array.
- **Dense Layer:** an embedded number of arbitrary neurons embedded in the layer where each neuron is a perceptron.

2.2 Techniques Brief:

2.2.1 Activation Function:

A mathematical operation that transforms the result or signals of neurons into a normalized output by introducing non-linearity within the network which enables the neural network to have greater representational power and solve complex functions.

- **ReLU Activation Function:** $y = \max(0, x)$. The function clamps down any negative values from the neuron to 0, and positive values remain unchanged.
- **Softmax Activation Function:** The function derives the probability distribution of a set of numbers within an input vector in which its set of values represents the probability of an occurrence of a class or event. The values within the vector all add up to 1

2.2.2 Dropout

Randomly reduces the number of interconnecting neurons within a neural network. At every training step, each neuron has a chance of being left out, or rather, dropped out of the collated contributions from connected neurons.

2.3 Calculations

Output tensor is calculated by:

$$w_{new} = \frac{w_o - filter\ size * 2 * padding + 1}{no.\ of\ strides}$$

Trainable parameters in Conv Layer = shape of width of the filter * shape of height of the filter * number of filters in the previous layer + 1 * number of filters

Trainable parameters in FC Layer = current layer neurons c * previous layer neurons $p+1$ * c

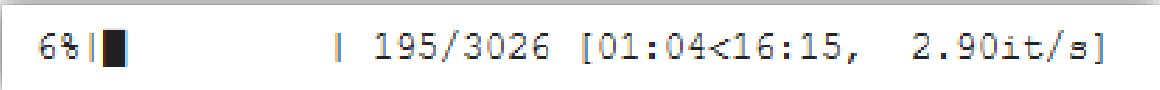
3. Algorithms

3.1 Keras Model

Typically, CNN architecture begins with feature extraction and finishes with classification. Feature extraction is performed by alternating convolution layers with pooling layers. Classification is performed with dense layers followed by a final softmax layer.

But before we start processing images, we need to perform some operations on the images first, or pre-processing. First, we read the images, and resize them to a smaller size (128x128 for the objects dataset). We found that the images have different sizes, and using their original sizes would use too much resources and crash the machine.

We used Google Colab for most of our work, but for some reason it would load the images much slower than a local machine would. So we used a library called **tqdm**(تقدم) that adds a progress bar to loops, that way we could monitor image loading and tell if something is wrong. Reading images took around 15 minutes on google colab.



```
6% | 195/3026 [01:04<16:15, 2.90it/s]
```

After reading and resizing the images, we normalize the images to be in the range of 0-1 by dividing by 255, this made the model's accuracy much better.

Now that our images are ready for training the model, we can go into the model's architecture.

We first started by using 2 **convolutional** layers with max pooling in between them, this resulted in getting low accuracy (92%), so we added more layers, which ended up being 3 **convolutional** layers, and 3 **max-pooling** layers. We tried adding more layers, but the accuracy would only drop down and overfitting would increase, meaning that the neural network was too complex for our images.

This model had acceptable accuracy, however, It was clear the mode was overfitting. To solve that, we added a **dropout** layer before the flattening layer.

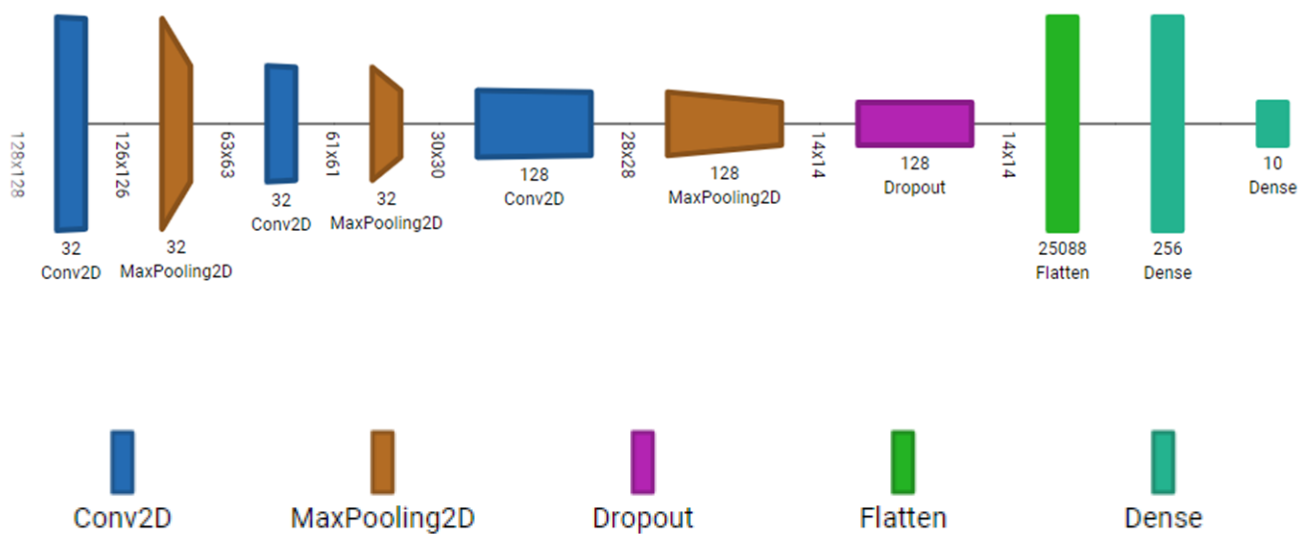
Finally, we have the fully-connected layer, then our softmax layer, which classifies into 10 classes.

The activation function used in all of the convolutional & dense layers is **Relu**, except the last layer is **softmax** so we can classify the images.

In choosing the optimization function, we tried AdaDelta, Adagrad, Adam, RMSProp and SGD. However, we found that the best optimization function to be **adam**, since other optimization functions would not reach the same accuracy, or would need far too many epochs.

The default learning rate (0.001) had good results, so we left it as is.

Below is the visualization of our convoluted neural network



We tried to use the same techniques on the cells dataset , but could only get ~70% accuracy , and after augmenting the data, we could reach ~85% accuracy, so we decided to focus our work on the caltech dataset.

Google Collab Specs & time elapsed

The first epoch always took the longest, 45 seconds, but after that:

Epochs: 25

Time per epoch: 2 seconds

Total Time: 1.5 minutes

Parameter	Google Colab
GPU	Nvidia K80 / T4
GPU Memory	12GB / 16GB
GPU Memory Clock	0.82GHz / 1.59GHz
Performance	4.1 TFLOPS / 8.1 TFLOPS
Support Mixed Precision	No / Yes
GPU Release Year	2014 / 2018
No. CPU Cores	2
Available RAM	12GB (upgradable to 26.75GB)

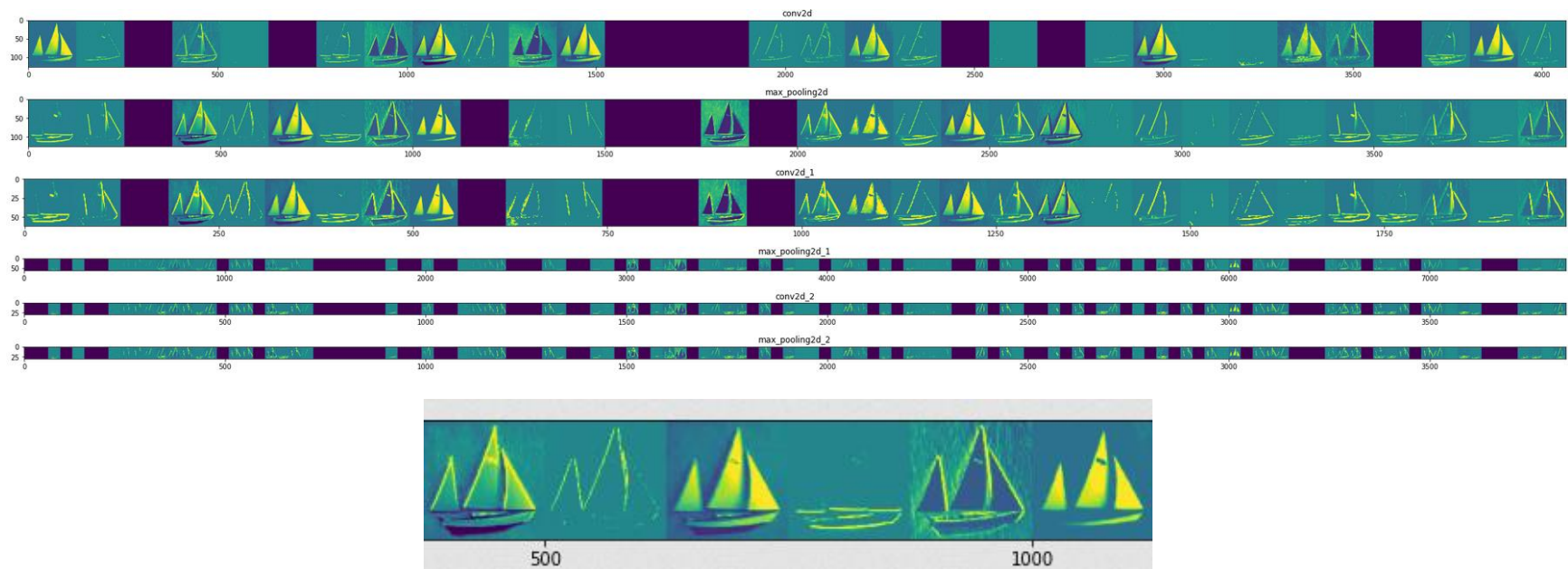
Visualizing the feature maps:

We can visualize the model's feature maps on any image. It shows how the different filters detect different features in the image. For example, we can see the features of this image of a sail boat (ketch). The model highlights features like diagonal lines of the sails, and the outline of the bottom part, they are the most prominent.

Original image:

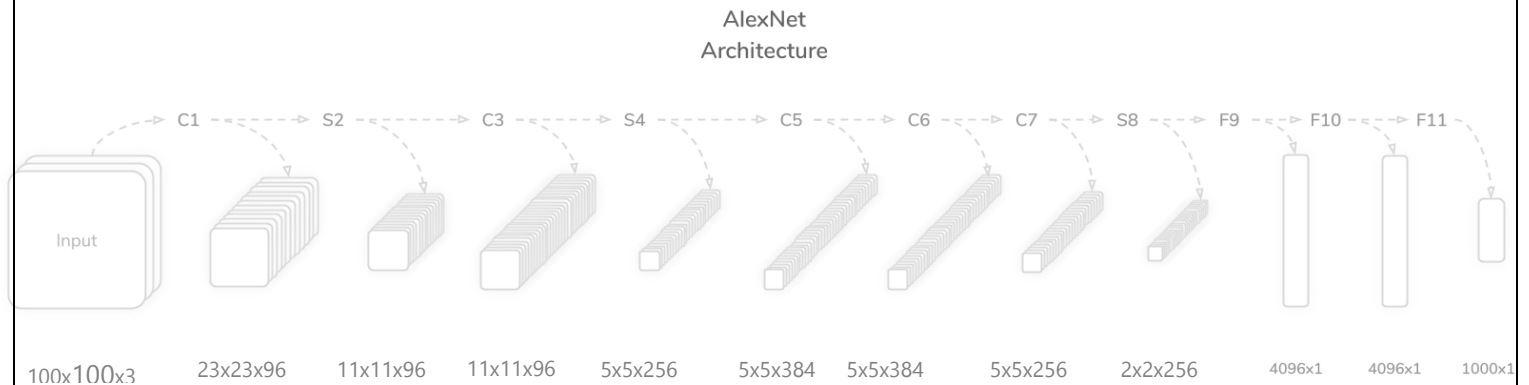


Feature maps:



3.2 Alex-Net:

Alex-Net contains 11 layers



First convolution layer: C1

Accepts a $(100 \times 100 \times 3)$ image. It performs a convolution operation using 96 filters with kernel size (11×11) with a stride of four and a padding of two. This produced a $(23 \times 23 \times 96)$ output tensor that was then passed through a ReLu activation function then on to the next layer. The layer contained 34,944 trainable parameters.

First Max Pooling Layer: S2

The second is a max-pooling layer that accepted output from the layer C1, a $(23 \times 23 \times 96)$ tensor, as its input. It performed a zero-padded sub-sampling operation using a (3×3) kernel with a stride of two. This produced a $(11 \times 11 \times 96)$ output tensor that was passed on to the next layer.

Second Convolution Layer: C3

The third layer is a convolutional layer that accepted output from the layer S2, a $(11 \times 11 \times 96)$ tensor, as its input. It performed a convolution operation using 256 (5×5) kernels with a stride of one, and a padding of two. This operation produced a $(11 \times 11 \times 256)$ output tensor that was then passed through a ReLu activation function, and then on to the next layer. The layer contained 614,656 trainable parameters, which summed to a total of 649,600 trainable parameters so far.

Second Max Pooling Layer: S4

The fourth layer of AlexNet is a max-pooling layer that accepted output from the layer C3, a $(11 \times 11 \times 256)$ tensor, as its input. It performed a zero-padded sub-sampling operation using a (3×3) kernel with a stride of two, similar to layer S2. This produced a $(5 \times 5 \times 256)$ output tensor that was then passed through a ReLU activation function, and then on to the next layer.

Third Convolution Layer: C5

The fifth layer of AlexNet is another convolutional layer that accepted output from the layer C5, a $(5 \times 5 \times 256)$ tensor, as its input. It performed a convolution operation using 384 (3×3) kernels with a stride and padding of one. This produced a $(5 \times 5 \times 384)$ output tensor that was then passed through a ReLU activation function, and then on to the next layer. The layer contained 885,120 trainable parameters, which summed to a total of 1,534,720 trainable parameters so far.

Fourth Convolution Layer: C6

The sixth layer of AlexNet is a convolutional layer that accepted output from the layer C5, a $(5 \times 5 \times 384)$ tensor, as its input. It performed the same convolution operation as layer C5, which lead to the same output size. The output was also passed through a ReLU activation function. The layer contained 1,327,488 trainable parameters, which summed to a total of 2,862,208 trainable parameters so far.

Fifth Convolution Layer: C7

The seventh layer of AlexNet was another convolutional layer that accepted a $(5 \times 5 \times 384)$ tensor from the layer C6 as its input. It performed a convolution operation using 256 (3×3) kernels with a stride and padding of 1. This produced a $(5 \times 5 \times 256)$ output tensor. The output was also passed through a ReLu activation function. The layer contained 884,992 trainable parameters, which summed to a total of 3,747,200 trainable parameters so far.

Third Max Pooling Layer: S8

The eighth layer is a max pooling layer that accepted a $(5 \times 5 \times 256)$ tensor from the layer C7 as its input. It performed a zero-padded sub-sampling operation using a (3×3) window region with a stride of two, similar to layer S2 and S4. This produced a $(2 \times 2 \times 256)$ output tensor that was then passed through a ReLU activation function, and then on to the next layer.

First Fully Connected Layer: F9

The ninth layer of is a fully connected layer that accepted a flattened $(2 \times 2 \times 256)$ tensor from the layer S8 as its input. It performed a weighted sum operation with an added bias term. This produced a (4096×1) output tensor that was then passed through a ReLu activation function, and on to the next layer. The layer contained 37,752,832 trainable parameters, which summed to a total of 41,500,032 trainable parameters so far.

Second Fully Connected Layer: F10

The tenth layer is another fully connected layer that accepted a (4096×1) tensor from the layer F9 as its input. It performed the same operation as layer F9 and produced the same (4096×1) output tensor that was then passed through a ReLu activation function, and then on to the next layer. The layer contained 16,781,312 trainable parameters, which summed to a total of 58,281,344 trainable parameters so far.

Third Fully Connected Layer: F11

The eleventh and final layer of the network is also a fully connected layer that accepted a (4096×1) tensor from the layer F10 as its input. It performed the same operation as layer F9 and F10 and produced a (1000×1) output tensor that was then passed through a softmax activation function. The layer contained 4,097,000 trainable parameters, which summed to a total of 62,378,344 trainable parameters overall. The output of the softmax activation function contained the predictions of the network.

4. Results and Discussion

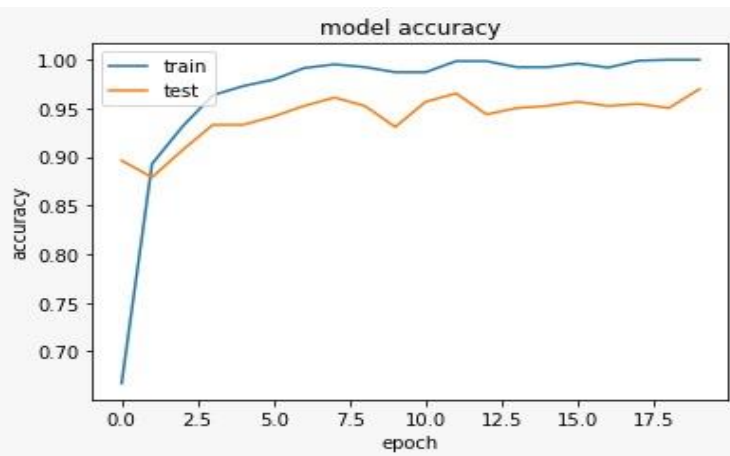
4.1 Keras Model

	precision	recall	f1-score	support
0	1.00	0.97	0.99	74
1	0.97	1.00	0.98	29
2	0.98	1.00	0.99	104
3	0.98	0.98	0.98	121
4	0.96	0.85	0.90	26
5	1.00	1.00	1.00	18
6	0.88	0.94	0.91	16
7	0.86	0.92	0.89	13
8	0.86	1.00	0.92	12
9	0.97	0.93	0.95	41
accuracy			0.97	454
macro avg	0.95	0.96	0.95	454
weighted avg	0.97	0.97	0.97	454

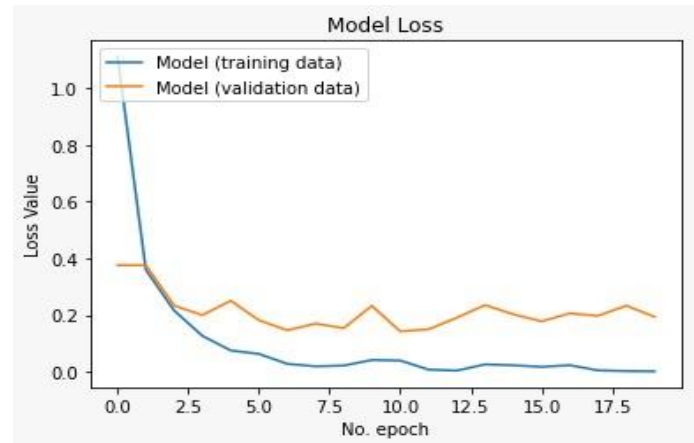
This is a summary of the results the model accuracy is approximately 97%, Very close to the alex-net model



Confusion matrix shows that revolver was the most confusing class for the model and it did predict all the car images right.



Model accuracy shows very good results and no overfitting



The loss values for alex-net were better but still they're acceptable here.

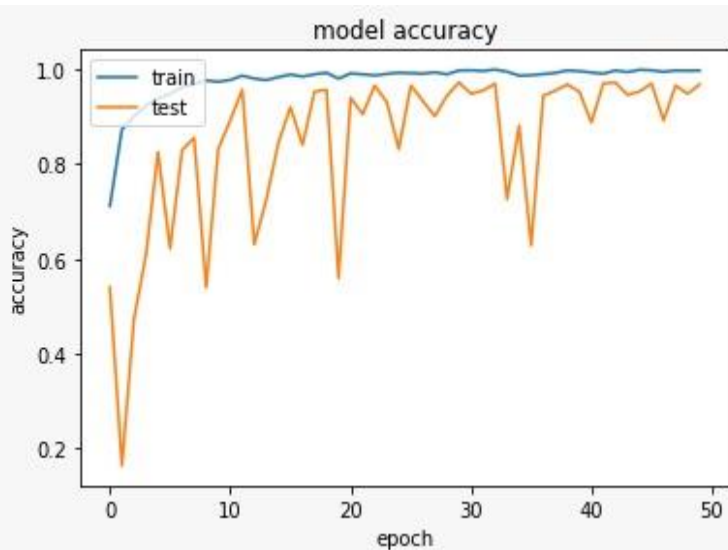
4.2 Alex-Net

	precision	recall	f1-score	support
0	1.00	0.99	0.99	72
1	0.97	1.00	0.99	38
2	0.98	1.00	0.99	103
3	1.00	0.98	0.99	123
4	0.88	0.83	0.86	18
5	1.00	1.00	1.00	21
6	0.73	0.92	0.81	12
7	0.96	1.00	0.98	23
8	0.80	0.80	0.80	5
9	0.94	0.87	0.91	39
accuracy			0.97	454
macro avg	0.93	0.94	0.93	454
weighted avg	0.97	0.97	0.97	454

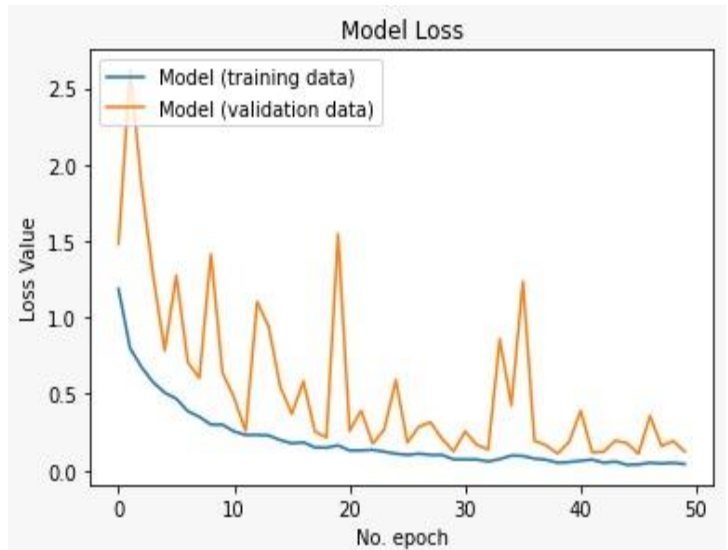
This is a summary of the results the model accuracy is approximately 97%

Confusion Matrix - Class Counts											
TRUE CLASS	watch			1		2	5	1	1	229	
	revolver						2		79	1	
	ketch							114			
	chandelier			1			101	4	1		
	car					122		1			
	bonsai	1	2	1		120		1	2	1	
	airplanes			2	796		1		1		
	Motorbikes			798							
	Leopards		200								
	Faces	434							1		
		Faces	Leopards	Motorbikes	airplanes	bonsai	car	chandelier	ketch	revolver	watch

Confusion matrix shows that watch was predicted as chandelier for 5 times and chandelier was predicted as ketch for 4 times.

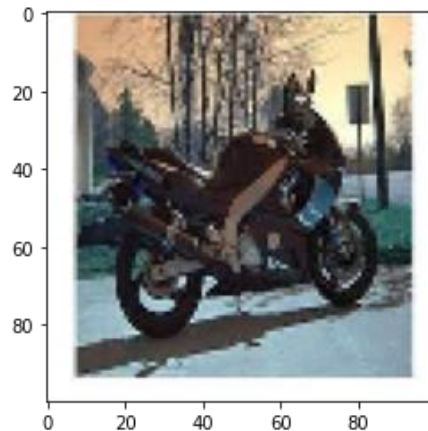
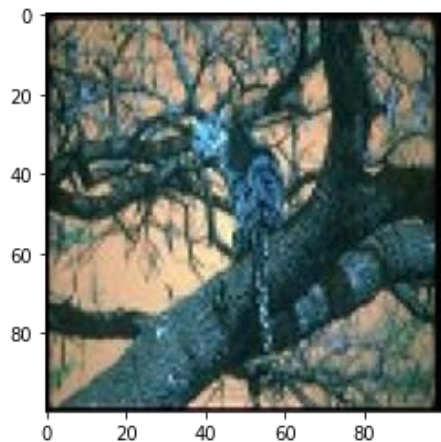


Model Accuracy shows almost no over-fitting



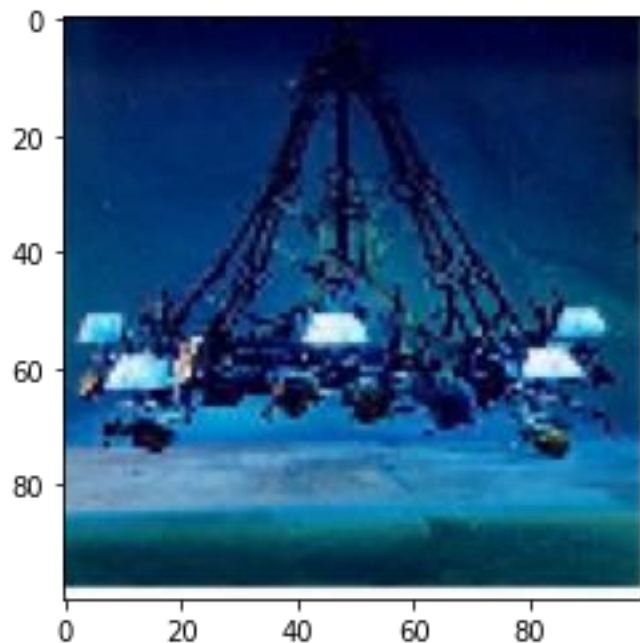
Model Loss shows that if we increased the number of epochs we might have got a better accuracy

4.3 Some examples for incorrect classification:

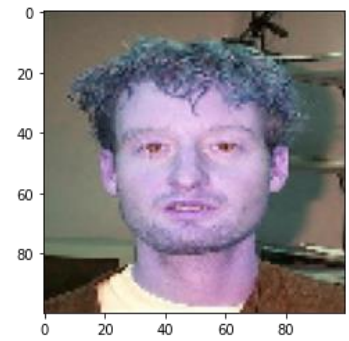
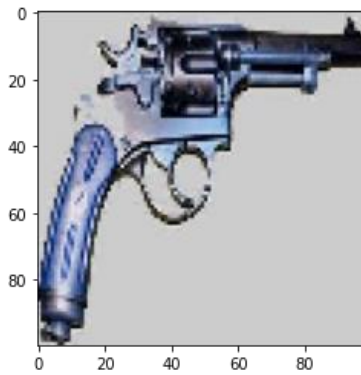
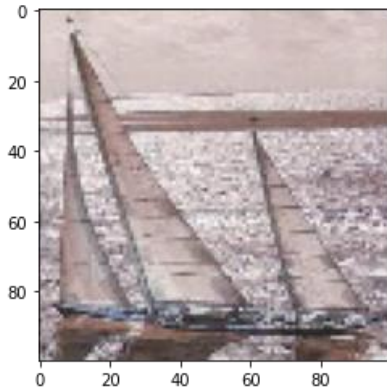


As we can see here in both of those pictures we have plants in the background which interestingly enough did confuse the model and had a wrong classification for those images as bonsai

This image was wrongly classified as an airplane in which the colors and the background looked like “sky” and confused the model.



4.4 Some examples for correct classification:



Appendix

Codes used

Keras Model

https://colab.research.google.com/drive/1LM0T7gg-HSs9Zm8bNoQOCygnZNs_llc8?usp=sharing

Alex-Net Model

<https://colab.research.google.com/drive/1KEwrTzkDup5UWnli2jEgWELmCj5nXv41?usp=sharing>

CSV

<https://drive.google.com/file/d/1x4QD5MIEQC4DG0ieqKWlyOdj1ffjIHdZ/view?usp=sharing>