Deep learning – Assignment 1

Image classification on CIFAR-100

Yonatan Zax - 204662779

We will start by naming some of the conclusion we have while writing the report.

Conclusions:

- **"Table of contents"** is a must while using any kind of notebook, might take time to write all the text cells, but the easy navigation is worth it at the end.
- **Data augmentation** was much more helpful than we expected at first, we will definitely use it in the future.
- **Printing some data samples** to inspect the augmentation changes and the resize changes were very helpful for understanding the benefit of using it.
- **Callbacks** are great time savers, by using EarlyStoping we were able to train our models with less unnecessary epochs and therefor spend less time waiting for the model's fitting process.
- **Neptune.ai** is a great way to manage our Experiments. At some point it's quite difficult to remember all the setting of your model, which one had better results. Neptune makes it easier to keep up with those experiments.

Table of Contents

0. Preparations:

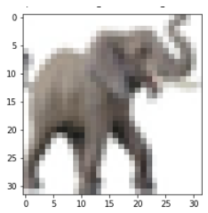
- Imports – from keras, sklearn and (pandas, numpy, matplotlib)
- Setting the Neptune account
- Load data – The data is available via keras.datasets

1. Data exploration

a. *Data size:* Train set has 50,000 records; Test set has 10,000 records.
Number of unique targets is 100

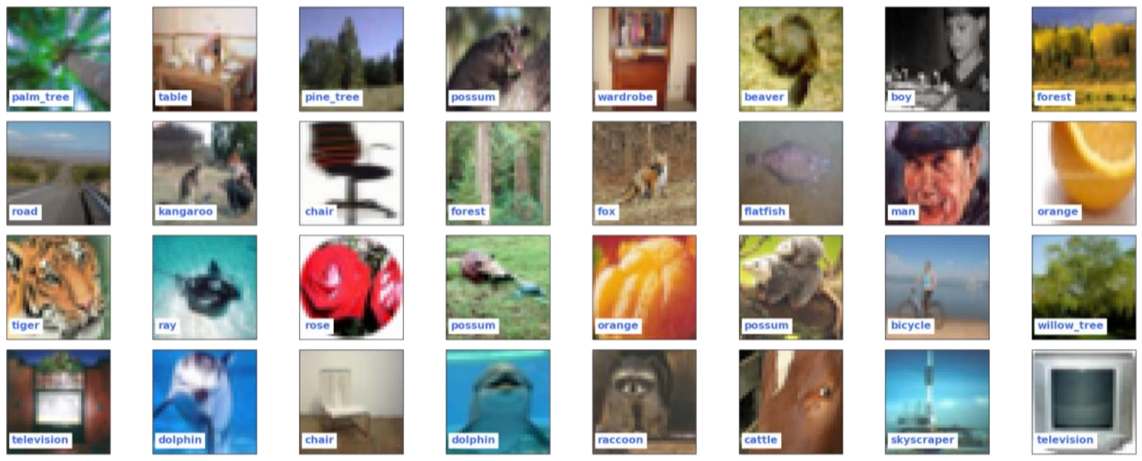
b. *What data does each sample contains:*

Each sample contains a single image and its classification.



Dimensions	Channels	Num of classes
(32x32) pixels	RGB (3)	100

<u>More Examples:</u>



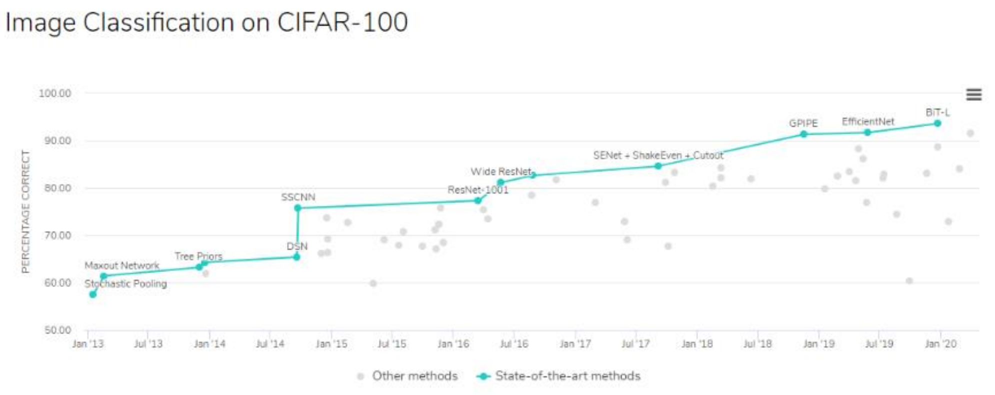
Preprocessing for the data? No, the data from keras.dataset is ready to use.
Augmentation: We can use augmentation, but the images are well cropped and the main element is centered therefor there is no need for augmentation for this dataset.

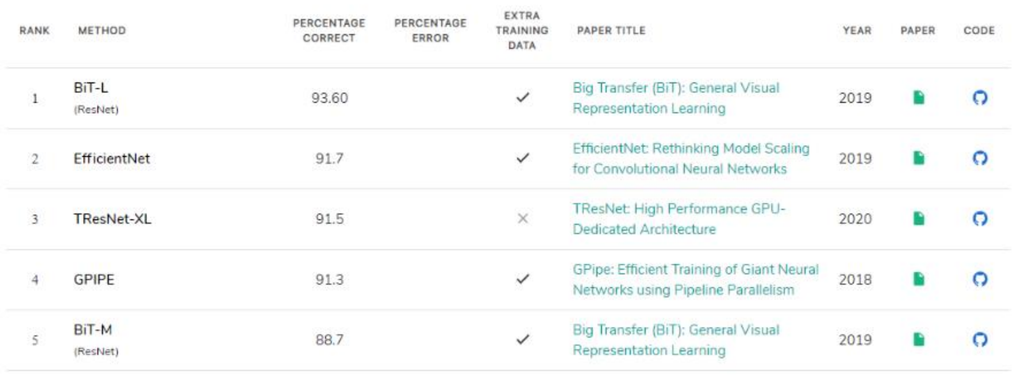
****Edit** – Later found out that using augmentation is also helpful and made a great usage as an improvement for the model.

c. *Is the data balanced?* Yes, the data is perfectly balanced - Each target has the exact same number of records (500 for the train, 100 for the test)

d. *Benchmark results on this dataset:*

Link to Benchmarks on CIFAR-100





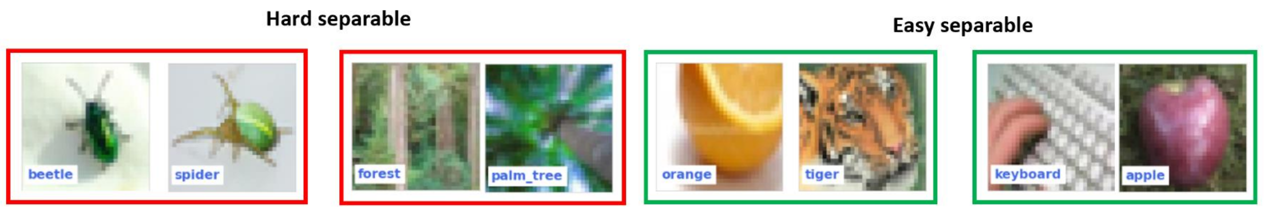
RANK	METHOD	PERCENTAGE CORRECT	PERCENTAGE ERROR	EXTRA TRAINING DATA	PAPER TITLE	YEAR	PAPER	CODE
1	BiT-L (ResNet)	93.60		✓	Big Transfer (BiT): General Visual Representation Learning	2019		
2	EfficientNet	91.7		✓	EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks	2019		
3	TResNet-XL	91.5		×	TResNet: High Performance GPU-Dedicated Architecture	2020		
4	GPIPE	91.3		✓	GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism	2018		
5	BiT-M (ResNet)	88.7		✓	Big Transfer (BiT): General Visual Representation Learning	2019		

We can see that the top 4 methods were able to get an accuracy score above 90%

Note that most of them were using 'Extra training data', this was not common in previous methods (from #6 downwards)

e. *Samples for each label (present easy and hard separable examples)*

Hard separable might be caused by similar backgrounds and\or colors of elements in the pictures, as well as similar shapes of the objects. Easy separable can be seen as images with different structures (colors, texture, backgrounds)



2. Building a DL model

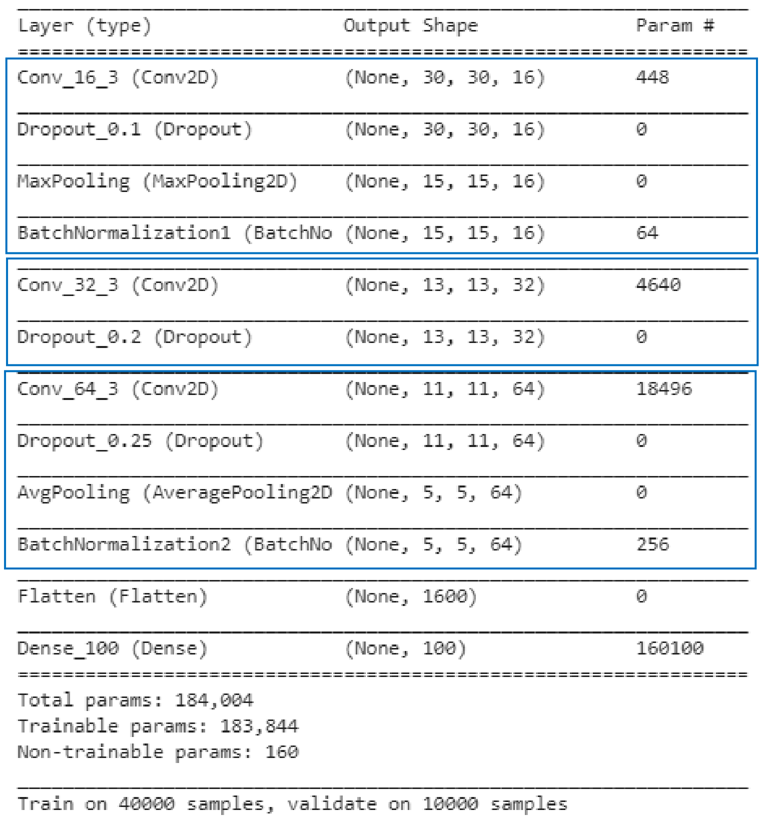
a. Validation strategy: We used validation_split of 0.2

Train size will be 80% (40,000 records), Test size is 20% (10,000 records)

b. Build model from scratch:

<u>Model's architecture:</u>

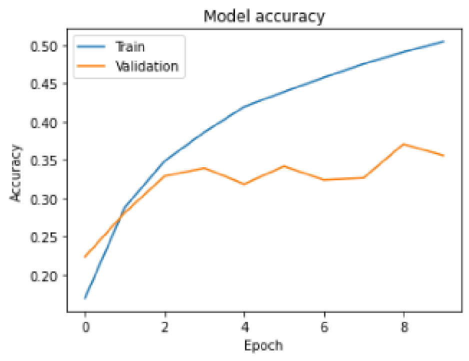
Our model contains 3 blocks of Conv2D + Dropout + Max\Avg pooling + BatchNormalization. All Conv2D layers are using 'relu' as activation function. The first Conv2D has 16 filters, the last has 64. The last layer is a Dense layer of 100 because we have 100 different classes.

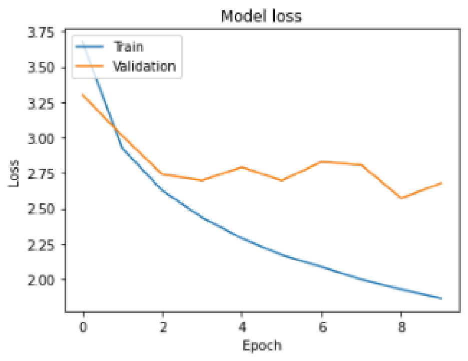


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Conv_16_3 (Conv2D)           (None, 30, 30, 16)        448
_________________________________________________________________
Dropout_0.1 (Dropout)        (None, 30, 30, 16)        0
_________________________________________________________________
MaxPooling (MaxPooling2D)    (None, 15, 15, 16)        0
_________________________________________________________________
BatchNormalization1 (BatchNo (None, 15, 15, 16)        64
_________________________________________________________________
Conv_32_3 (Conv2D)           (None, 13, 13, 32)        4640
_________________________________________________________________
Dropout_0.2 (Dropout)        (None, 13, 13, 32)        0
_________________________________________________________________
Conv_64_3 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
Dropout_0.25 (Dropout)       (None, 11, 11, 64)        0
_________________________________________________________________
AvgPooling (AveragePooling2D (None, 5, 5, 64)          0
_________________________________________________________________
BatchNormalization2 (BatchNo (None, 5, 5, 64)          256
_________________________________________________________________
Flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
Dense_100 (Dense)            (None, 100)               160100
=================================================================
Total params: 184,004
Trainable params: 183,844
Non-trainable params: 160
_________________________________________________________________
Train on 40000 samples, validate on 10000 samples
```

<u>Training metrics:</u>

```
-----------    Training score
Accuracy = 0.50
Loss = 1.86


-----------    Validation score
Validation Accuracy = 0.36
Validation Loss = 2.68


-----------
```



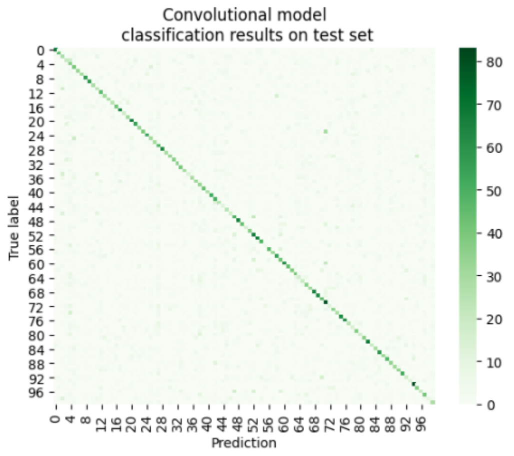


Evaluate the results:

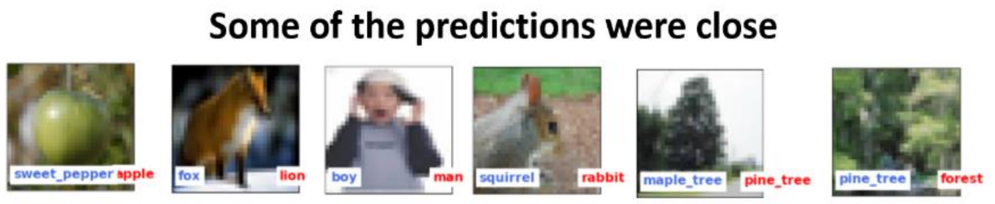
There is a big difference between the train and validation caused by overfitting. As we can see, the Accuracy on the Train set is getting better each epoch while the validation is stabilized around 0.35. Same for the Loss, Train set is improving while the validation doesn't.

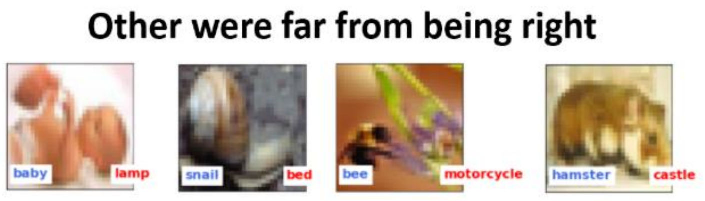
Basic model's prediction: Model's accuracy on test set is 38.7%

Confusion matrix: High values are located only on the diagonal line, meaning that there are no alternative classifications. Example: If there was high prediction outside of the diagonal line, it's a sign that the model is classifying one of the label as a different one. In our case, the model misses predictions over all targets.



Analyzing wrong predictions:





Even humans might find it difficult to classify the first row. Trees look alike and it's hard to get the correct one (same for Boy-Man and Squirrel-Rabbit)

The second row doesn't have any elements that might be similar between expected and actual prediction.

Comparing test set accuracy with validation accuracy

Validation	**Test set**
36.0 %	38.7 %

c. How to improve?

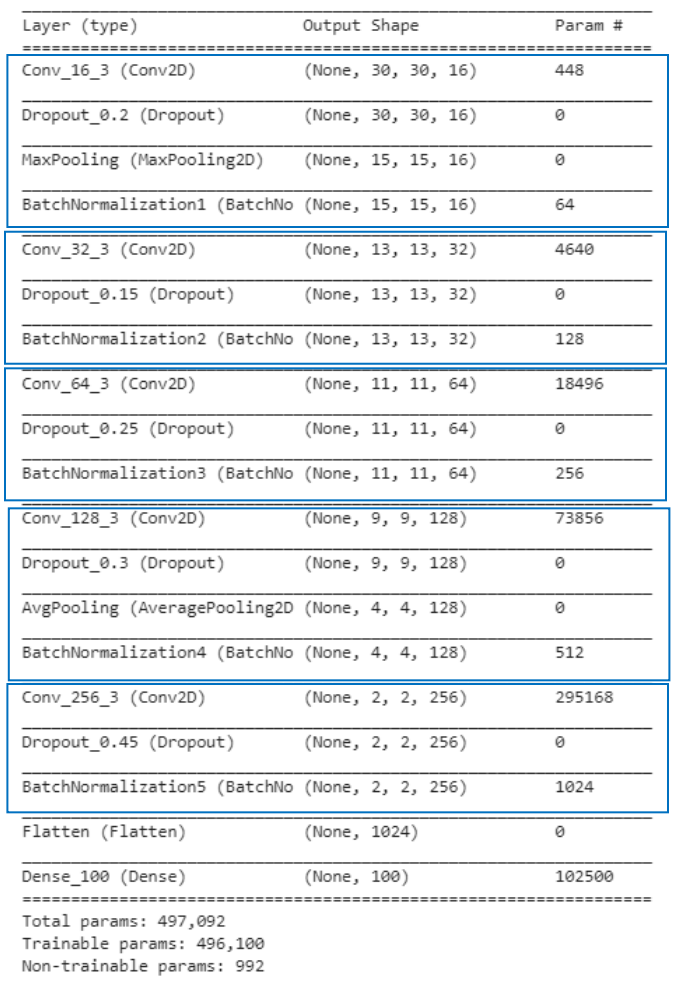
1. Build Conv2D layers up to size 256 (instead of 64) – Get a more complex model.
2. Use more epochs (instead of 10) – Let the model train more
3. Normalize the data, using X_train.astype('float32')/255

At first, we will show improves 1-3, later 4-5 will be added as well on another model.

4. Use Augmentation – Get different a variety of samples from a single image.
5. Add more data to training - Changing the split-validate ratio

d. Build an Improved model

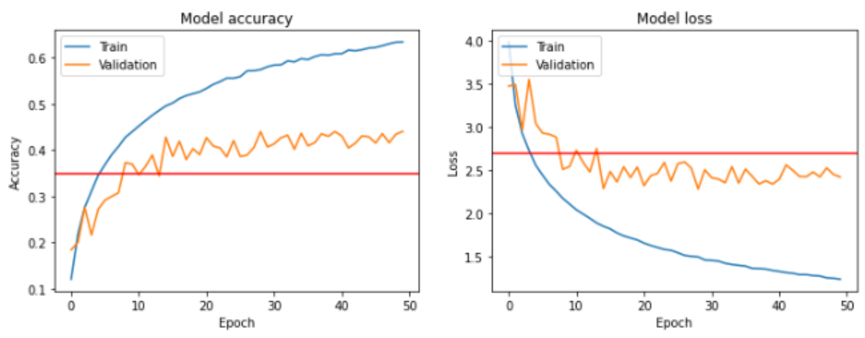
Model's architecture:
Our model contains 5 blocks of Conv2D + Dropout + Max\Avg pooling + BatchNormalization. All Conv2D layers are using 'relu' as activation function. The first Conv2D has 16 filters, the last has 256. The last layer is a Dense layer of 100 because we have 100 different classes.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Conv_16_3 (Conv2D)           (None, 30, 30, 16)        448
_________________________________________________________________
Dropout_0.2 (Dropout)        (None, 30, 30, 16)        0
_________________________________________________________________
MaxPooling (MaxPooling2D)    (None, 15, 15, 16)        0
_________________________________________________________________
BatchNormalization1 (BatchNo (None, 15, 15, 16)        64
_________________________________________________________________
Conv_32_3 (Conv2D)           (None, 13, 13, 32)        4640
_________________________________________________________________
Dropout_0.15 (Dropout)       (None, 13, 13, 32)        0
_________________________________________________________________
BatchNormalization2 (BatchNo (None, 13, 13, 32)        128
_________________________________________________________________
Conv_64_3 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
Dropout_0.25 (Dropout)       (None, 11, 11, 64)        0
_________________________________________________________________
BatchNormalization3 (BatchNo (None, 11, 11, 64)        256
_________________________________________________________________
Conv_128_3 (Conv2D)          (None, 9, 9, 128)         73856
_________________________________________________________________
Dropout_0.3 (Dropout)        (None, 9, 9, 128)         0
_________________________________________________________________
AvgPooling (AveragePooling2D (None, 4, 4, 128)         0
_________________________________________________________________
BatchNormalization4 (BatchNo (None, 4, 4, 128)         512
_________________________________________________________________
Conv_256_3 (Conv2D)          (None, 2, 2, 256)         295168
_________________________________________________________________
Dropout_0.45 (Dropout)       (None, 2, 2, 256)         0
_________________________________________________________________
BatchNormalization5 (BatchNo (None, 2, 2, 256)         1024
_________________________________________________________________
Flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
Dense_100 (Dense)            (None, 100)               102500
=================================================================
Total params: 497,092
Trainable params: 496,100
Non-trainable params: 992
```

Training metrics:
Due to the improves mentioned above, we got better results. Previous validation accuracy was 0.36. But, as seen in the graph below, the model is still overfitting.

```
-----------      Training score
Accuracy = 0.63
Loss = 1.24


-----------      Validation score
Validation Accuracy = 0.44
Validation Loss = 2.42
```



Basic model's prediction: Model's accuracy on test set is 43.93%
Result on testset is exactly as the validation accuracy.

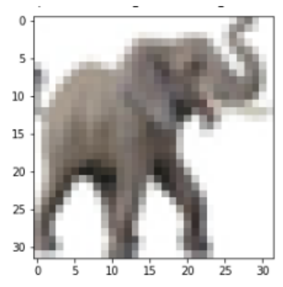
Improved model Version$_2$ (Using augmentation):

In order to overcome the overfitting, we will try to add more samples to the training data, did this by setting the validation_split to 0.9
In addition, we add ImageDataGenerator for augmentation usage. This way every image will be generated to a variety on samples at each epoch.

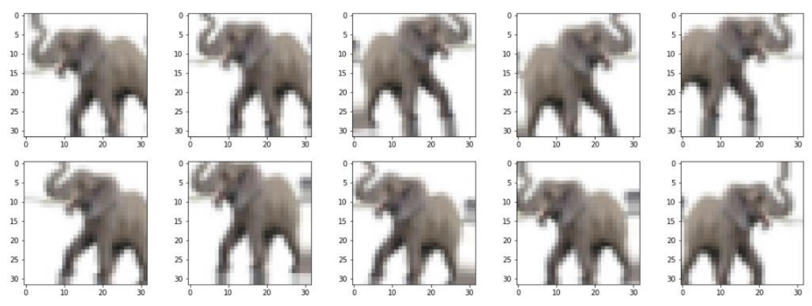
Augmentation parameters

1. rotation_range = 10 (Image will rotate at most 10 deg)
2. width_shift_range = 0.1 (Image will shift horizontally)
3. height_shift_range = 0.1 (Image will shift vertically)
4. shear_range = 0.01 (Shear angle in counter-clockwise direction in degrees)
5. zoom_range = 0.05 (Image will be zoomed-in)
6. horizontal_flip = True (Allowing horizontal flip)
7. vertical_flip = False (Vertical flip is not allowed)

Original

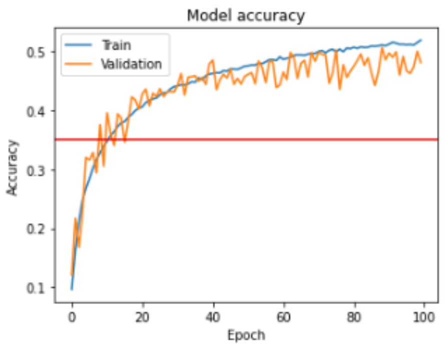


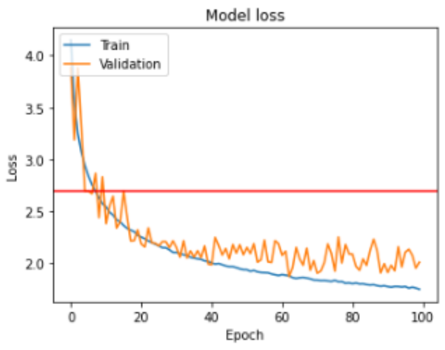
Examples from ImageDataGenerator



<u>Training metrics:</u>
Due to more training samples we can see that both training and validation are more correlated. Also, the validation accuracy is higher than before.

```
-----------    Training score
Accuracy = 0.52
Loss = 1.75


-----------    Validation score
Validation Accuracy = 0.48
Validation Loss = 2.01
```

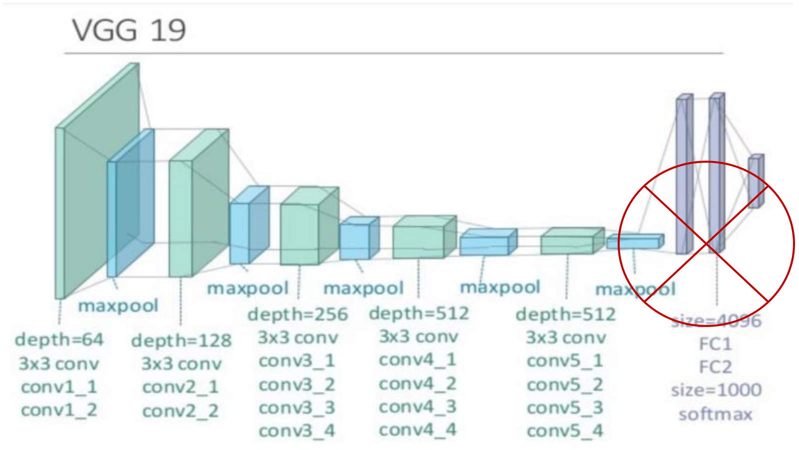




<u>Basic model's prediction:</u> Model's accuracy on test set is 48.76%
Result on testset is exactly as the validation accuracy.

3. Trained model architecture

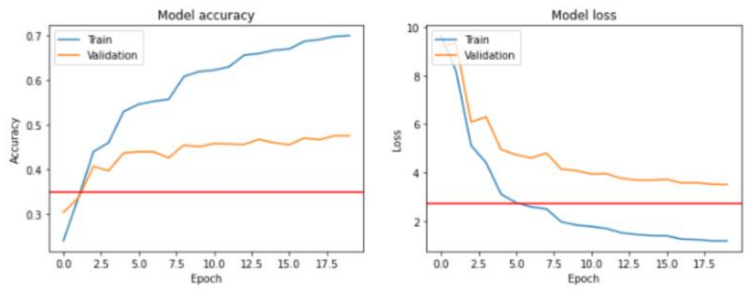
We used the VGG-19 model with “imagenet” weights. This model is well known and was presented in our last lecturer.
At first the model wasn’t doing so well, after consulting with a friend who used this model before we were advised to drop the last 6 layers of the VGG-19 model.



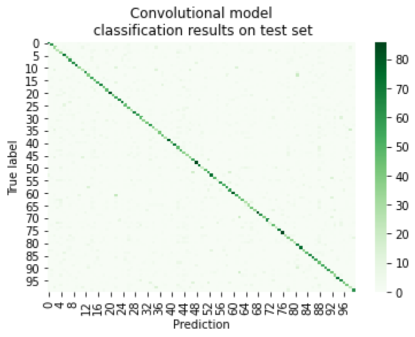
a. *Change the last layer:*

We’ve added a flatten layer followed by a Dense output layer with 100 neurons.
We used 20 epochs for fitting.

Evaluate the results



As we can see, the model is over fitting.
<u>Confusion matrix:</u> Accuracy on test set is 51.2%



d. Use the trained model as a feature extraction:

We used Random Forest Classifier, Accuracy on test set is 40%

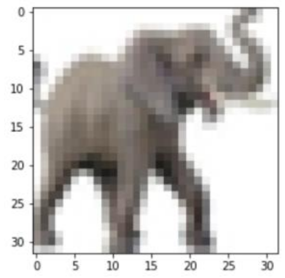
e. Preform model stacking to improve performance

Improves

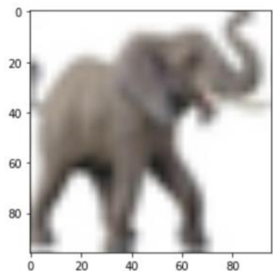
1. Increase the image size (Due to RAM limit, we will increase to 96x96)
2. Add layers to the model
   - Add Dropout layer to handle overfitting
   - Add BatchNormalization layers
3. Fit using more epochs

Resize image example:

32x32



96x96



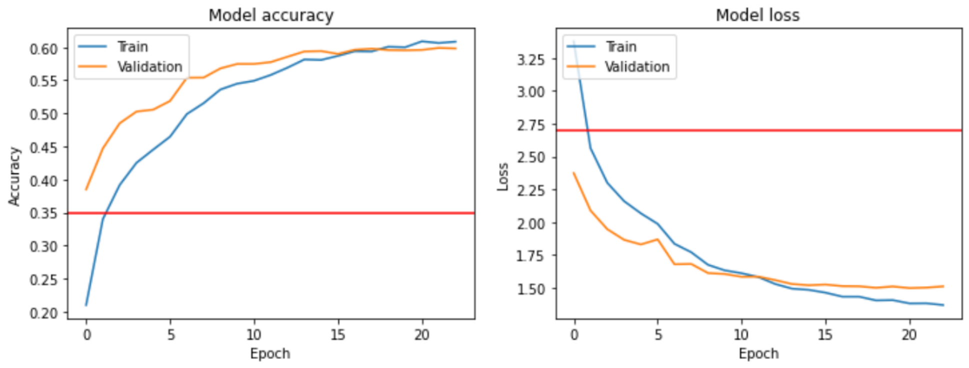
Added layers on top of the VGG-19 architecture.

```
_________________________________________________________________
Flatten (Flatten)            (None, 73728)             0
_________________________________________________________________
Dense_512 (Dense)            (None, 512)               37749248
_________________________________________________________________
BatchNormalization (BatchNor (None, 512)               2048
_________________________________________________________________
Dropout_0.3 (Dropout)        (None, 512)               0
_________________________________________________________________
Dense_256 (Dense)            (None, 256)               131328
_________________________________________________________________
Dropout_0.4 (Dropout)        (None, 256)               0
_________________________________________________________________
dense_7 (Dense)              (None, 100)               25700
=================================================================
Total params: 48,493,476
Trainable params: 37,907,300
Non-trainable params: 10,586,176
```

By using EarlyStoping, the model fitting was over after 23/50 epochs.

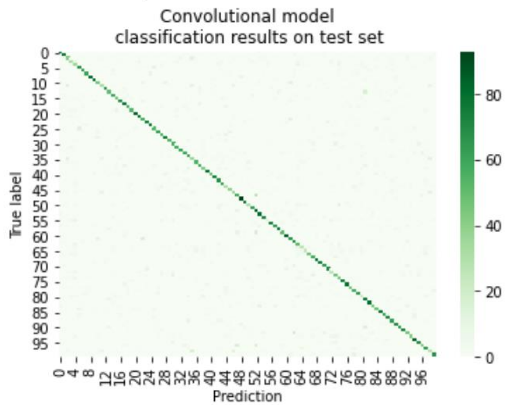
```
     -----------    Training score    -----------
Accuracy = 0.61
Loss = 1.37

    -----------    Validation score    -----------
Validation Accuracy = 0.60
Validation Loss = 1.51
```

Evaluate the results:



The model has good fit, validation score is very similar to the training.
Moreover, the Accuracy we got (60%) is better that the model from scratch from section 2 (48%).
Confusion matrix

```
model accuracy on test set is: 59.48%
```



The accuracy on the test set is similar to the validation accuracy (60%)

Use as feature extractor:

We will use Random Forest Classifier, Accuracy on test set is 57%.
A significant improve than the previous model (40%)

*** Reminder – Conclusions are shown in the first page ***

Finally, hoping that you got this far, this assignment was indeed very long.
We've learned a lot about image classification, approaches to improve accuracy and manage overfitting. We feel more confident about our understanding of the pipeline, from data exploration through preprocessing on to training a model and examining the results.