MSc in Business Analytics
Course: Social Media Analytics
Associate Professor G. Papagewrgiou
Supervisor: D. Pappas

TENSORFLOW ASSIGNMENT
GROUP ASSIGNMENT

CHATZIMOSCHOU ANGELIKI
BAPT1534
MELENIKOU GALATEIA
BAPT1527

MAY 2017

TABLE OF CONTENTS

PART I – DATA REPRESANTATION AND EXPLORATION

In this project, we are assigned to demonstrate the capabilities of deep learning techniques, in the domain of visual recognition. We deploy Convolutional Neural Networks on the "CIFAR-10" dataset provided by University of Toronto. It is a standard computer vision and deep learning data set developed by the Canadian Institute for Advanced Research (CIFAR).

Specifically, this dataset consists of 60.000 images of size 32x32 pixel squares in RGB format (thus 32x32x3: height, width, depth), divided into 10 classes, with 6.000 images per class. There are 50.000 training images and 10.000 test images. Additionally, it is divided into 5 training batches and 1 test batch, each with 10.000 images. The test batch contains exactly 1.000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5.000 images from each class.

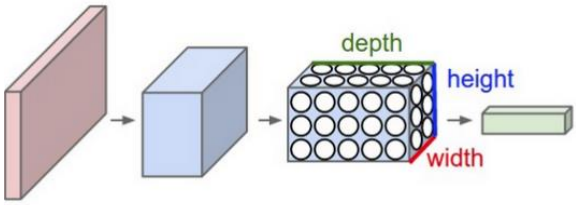


Figure 1 - Image height, width, depth

The goal is to classify the RGB images across the following 10 categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

Figure 2 – Cifar10 categories

We are going to build several Convolutional Neural Networks (CNNs) models for classifying the images in question dataset.

PART II – DATA PREPROCESSING & DATA AUGMENTATION

There are several techniques of image pre-processing, such as: mean image subtraction, per-channel normalization (subtract mean, divide by standard deviation), per-channel mean subtraction, Dimensionality Reduction (ZCA). The easiest are often the most effective and mainly we want to distribute it as widely as possible.

When working with limited sized datasets, it is highly possible to overfit, thus it is generally a good strategy to augment it and help our model generalize better. Data augmentation, which contains several techniques (random flip, random rotation, zooming, cropping, whitening -turn the distribution into a normal distribution), helps us to generate artificially more images by randomly transforming the features of our initial ones and expands our initial dataset.

Data augmentation works as follows: at each learning epoch, transformations with randomly selected parameters, within the specified range are applied to all original images in the training set. After an epoch is completed, i.e. after having exposed a learning algorithm to the entire set of training data, the next learning epoch is started and training data is once again augmented, by applying specified transformations to the original training data. In this way the learning algorithm almost never sees two exactly the same training examples, because at each epoch training examples are randomly transformed.

In this project, we apply mean subtraction (to get them zero-centered) and standard normalization pre-processing methods, so as to bring our input features at a similar scale and make them be of same importance.

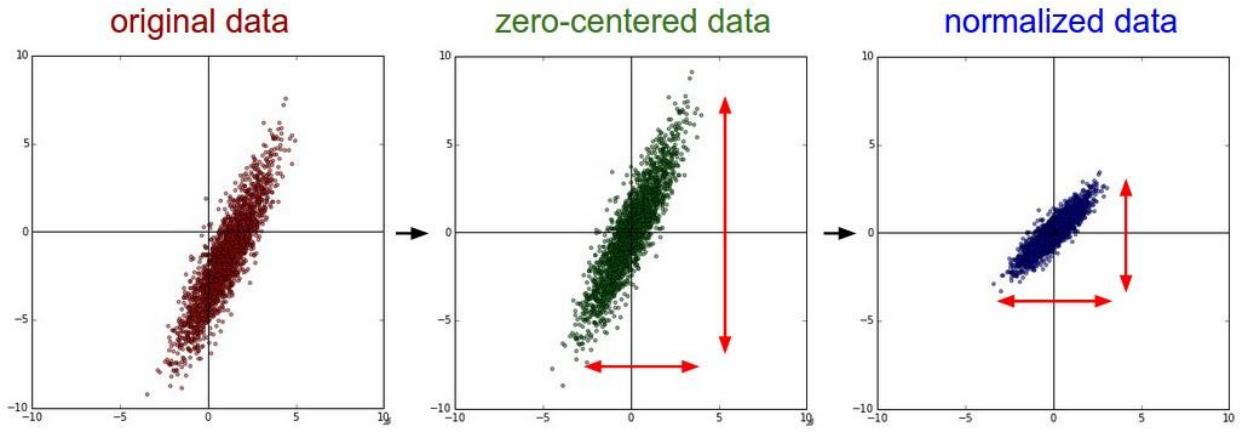
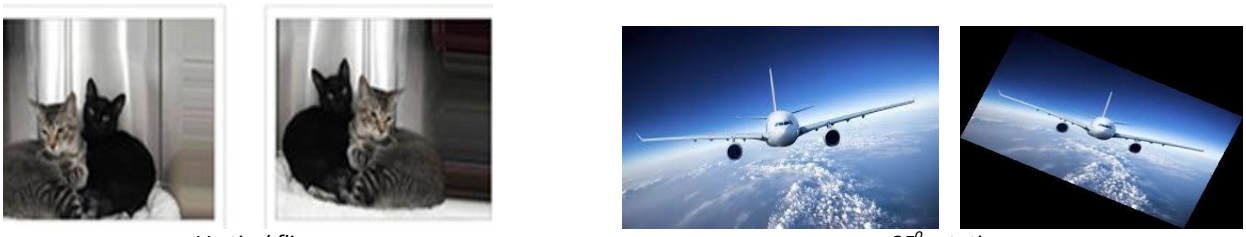


Figure 3 – Data preprocessing

In addition, to produce more images, we will slightly transform our original photos by flipping them vertically and rotating them by 25^{o}.



Vertical flip *25^{o} rotation*

Figure 4 – Data preprocessing

PART III – CNN ARCHITECTURE

CONVOLUTIONAL LAYERS

Convolution layers are another functional element that we can add to Neural Networks and they are widely used in visual recognition. They work similarly to how blur filters work on images by applying the Gaussian blurring technique. An *nxn* square box ('moving filter') passes through every pixel of the original image, and a new pixel color is produced by averaging all the pixels in the box. Each pixel in the filter is weighted based on how far away it is from the center. Furthermore, pixels closer to the center affect the output more than those farther away. This leads to a more natural blurring. In CNNs the weights are trainable parameters, using the optimizer of our choice (Adam, Adagrad, Adadelta, Stochastic Gradient Descent, etc.) iteratively, improving the performance of the entire network.

Each Convolutional Layer deconstructs the image into different repetitive patterns, in order to provide different information about the image. This way, the newly produced features of the initial image can be used, so as to classify it more effectively. The more complicated our model is, which means the more Convolutional Layers we add, the more its parameters (weights, biases, etc.) will increase. This engulfs the danger of overfitting and also longer training times.

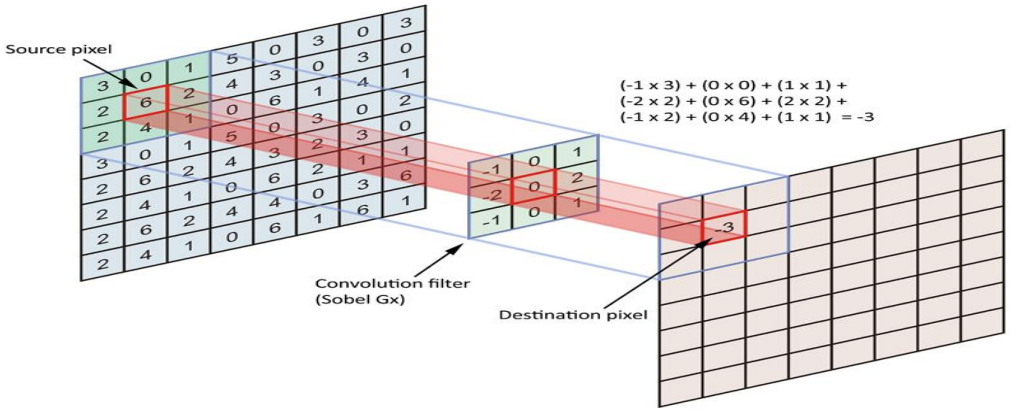


Figure 5 – Convolutional Layer

ACTIVATION FUNCTIONS

Activation functions introduce linear or non-linear properties to the Convolutional Neural Network. Their main purpose is to convert an input signal of a node in a CNN model to an output signal, which itself becomes input of the next layer in the stack. There are several non-linear activation functions, such as: Sigmoid, Hyperbolic Tangent, ReLu, etc.

An important feature of an Activation function is that it should be differentiable. We need it to be this way, so as to perform backpropagation optimization strategy to compute gradients of error (loss) with respect to weights. Then, accordingly, optimize weights, using Gradient descend or any other optimization technique to reduce error. In addition, when no activation function is performed, this transforms the neural network into a simple linear regression model and also it wouldn't be able to learn complicated data structures like images, videos, audios, speech etc.

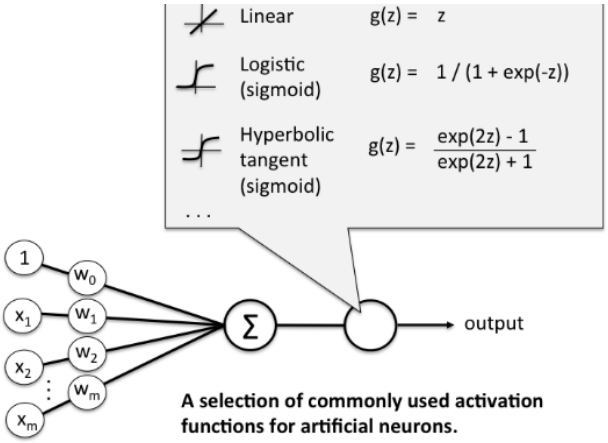


Figure 6 – Activation Functions

We implement Rectified Linear Units (ReLu) activation function on our models, as it is the mostly recommended function for CNNs. Additionally, it avoids and corrects the vanishing gradient problem and it is 6 times better in convergence than Tanh function. The only constraint is that it applies only in the hidden layers of our CNN. Unfortunately, one problem with ReLu is that some gradients can be fragile during training and may lead to dead neurons. To fix this problem, another modification was introduced called Leaky ReLu.

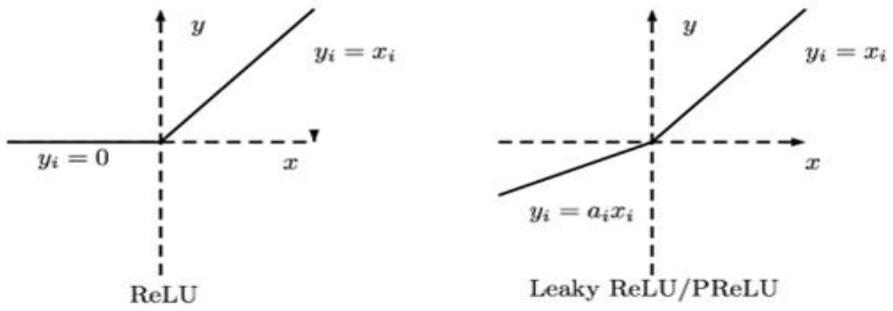


Figure 7 – ReLu Functions

Finally, for the output layer we use a Softmax function, in order to compute the probabilities of the classes.

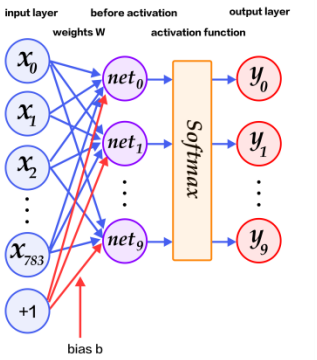


Figure 8 – Softmax Function

MAX POOLING

Since our images are in RGB format, the output of the convolutional layers will be a 4D tensor. We want to reduce the size of our image vector (down-sampling), in order to reduce at the same time the complexity of our model and the training time it requires. To achieve this, we perform a down-sampling technique called Max-Pooling, which divides the image into non-overlapping sections and picks the largest value of each section as the new value of the section. Here, we implemented Max-Pooling layers of 2x2 pool-sized regions (Strides).

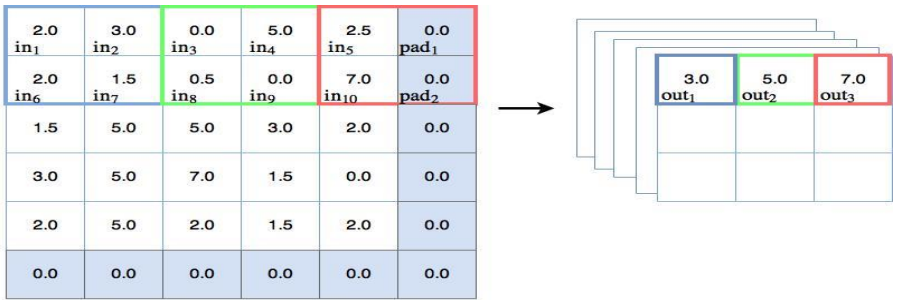


Figure 9 – Max Pooling

STRIDES

In the Convolutional / 'Moving' Window, the 3×3 filter (in our project) moves only a single place in the x and y direction through the image / input. This is called a Stride of (1, 1), which means that the filter moves 1 step in the x and y directions. With Max-Pooling, the Stride is usually set so that there is no overlap between the regions. In this case, we need a Stride of (2, 2) to reduce dimensionality and avoid overlap.

PADDING

Padding nodes are dummy nodes that are inserted so that the 2×2 max pooling filter can make 3 steps in the x and y directions with a stride of 2, despite having an even number of nodes to traverse in either the x or y directions.

In our models, we deploy Zero-Padding which refers to the process of symmetrically adding zeroes to the input matrix. It's a commonly used modification, which allows the size of the input to be adjusted to our requirement. It is mostly used in designing the CNN layers when the dimensions of the input volume need to be preserved in the output volume. These nodes will never actually be selected in the Max-Pooling process as their values are 0.

0	0	0	0	0	0	0	0	0	0
0	77	80	82	78	70	82	82	140	0
0	83	78	80	83	82	77	94	151	0
0	87	82	81	80	74	75	112	152	0
0	87	87	85	77	66	99	151	167	0
0	84	79	77	78	76	107	162	160	0
0	86	72	70	72	81	151	166	151	0
0	78	72	73	73	107	166	170	148	0
0	76	76	77	84	147	180	168	142	0
0	0	0	0	0	0	0	0	0	0

Figure 10 – Zero - padding

DROPOUT

Another technique to diminish the possibility of overfitting is Dropout. It is a regularization technique, by which some Neurons are thrown away (keeping only other, more useful neurons), depending on their contribution to the final output. It prevents a layer from seeing twice the exact same pattern and eliminates random data correlations. Here we use both an aggressive Dropout strategy by throwing away half of the Neurons (Dropout=0.5), as well as a less aggressive one which throws away just a quarter of the Neurons (Dropout=0.25).

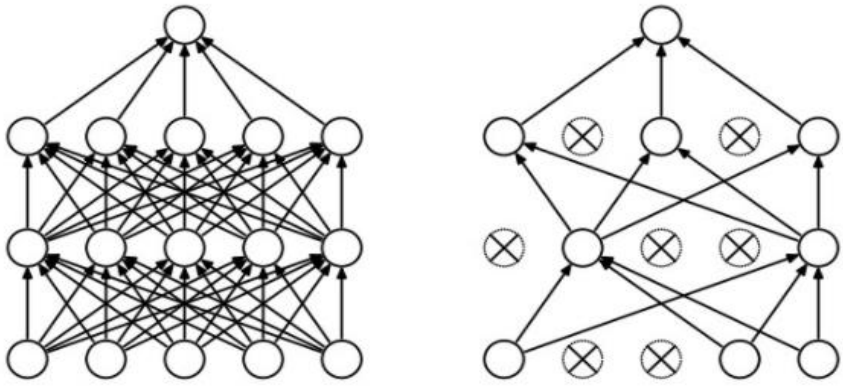


Figure 11 – Dropout

FULLY CONNECTED LAYERS

The Fully-Connected Layers come at the end of a Convolutional Neural Network model in order to complete the classification process of our input photos. To connect the output of the final Max-Pooling layer to the Fully-Connected Layer, we have to flatten its output into a single *(N x 1)* tensor.

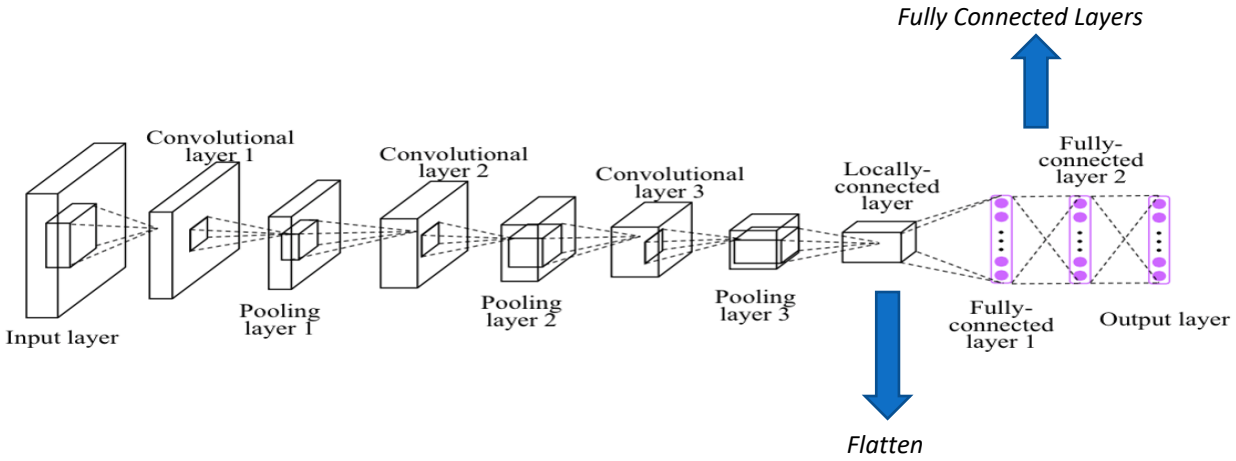


Figure 12 – Dropout CNN architecture example

PART IV – MODELS DEPLOYED

In order to build our models we used Tensorflow, a Python library with lots of power and flexibility. However, since Tensorflow can be a bit verbose for quick prototyping, we also used Keras which is a high level library and operates over Tensorflow.

Keras offers the option to use the Sequential structure to build our models. By using the Sequential form, the layers are getting stacked linearly and in order (from input to output). While constructing the model, we should specify the input shape, only at first layer, because the following layers do automatic shape inference. More information about the model structure is given in each model described respectively below.

In all our model implementations, before the training phase, we compiled our model using the following:

1. Loss function (the objective that the model tries to minimize each time):
   We use the standard cross entropy for categorical classification.
2. Optimizer: we chose Adam optimizer, an improved algorithm of SGD for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments.
3. Metrics (the function used to judge the performance of our model): Since, it is a classification problem we decided to use accuracy.

Finally, Keras models are trained on Numpy arrays of input data and labels.

1ST MULTILAYER MODEL FOR 2 EPOCHS

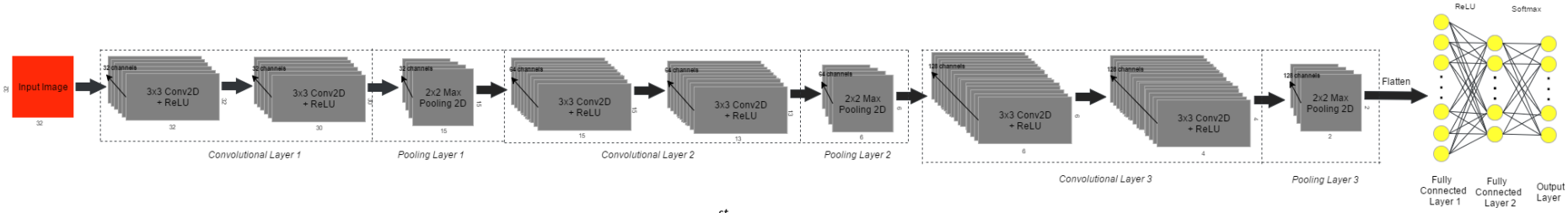


Figure 13 – Our 1st Multilayer Model

For starters, our best applied method, in terms of accuracy and loss, is a multi-layer model architecture consisting of convolutions and nonlinearities. These layers are followed by fully connected layers leading into a softmax classifier. The detailed structure is given in the table below. Concerning the pre-processing phase, we only deployed image normalization.

Layer Name	Type	Filter size/ stride	Output size	Parameters
Conv2d_1	Convolutional	3x3 / 1	32x32x32	896
Activation_1	Activation		32x32x32	0
Conv2d_2	Convolutional	3x3 / 1	30x30x32	9248
Activation_2	Activation		30x30x32	0
Max_pooling2d_1	Max Pooling		15x15x32	0
Dropout_1	Dropout (25%)		15x15x32	0
Conv2d_3	Convolutional	3x3 / 1	15x15x64	18496
Activation_3	Activation		15x15x64	0
Conv2d_4	Convolutional		13x13x64	36928
Activation_4	Activation		13x13x64	0
Max_pooling2d_2	Max Pooling		6x6x64	0
Dropout_2	Dropout (25%)		6x6x64	0
Conv2d_5	Convolutional		6x6x128	73856
Activation_5	Activation		6x6x128	0

Conv2d_6	Convolutional	4x4x128	147584
Activation_6	Activation	4x4x128	0
Max_pooling2d_3	Max Pooling	2x2x128	0
Dropout_3	Dropout (25%)	2x2x128	0
Flatten_1	Flatten/Reshape	512	
Dense_1	Fully Connected	512	512 (input values) * 512 (neurons) + 512 (bias values) = 262656
Activation_7	Activation	512	0
Dropout_4	Dropout (50%)	512	0
Dense_2	Fully Connected	256	512 (input values) * 256 (neurons) + 256 (bias values) = 131328
Activation_8	Activation	256	0
Dropout_5	Dropout (50%)	256	0
Logits/Cost (dense_3)	Softmax	10	[256 (input weights) +1 (bias weight)] * 10 classes = 2570
		Total/Trainable params:	683562

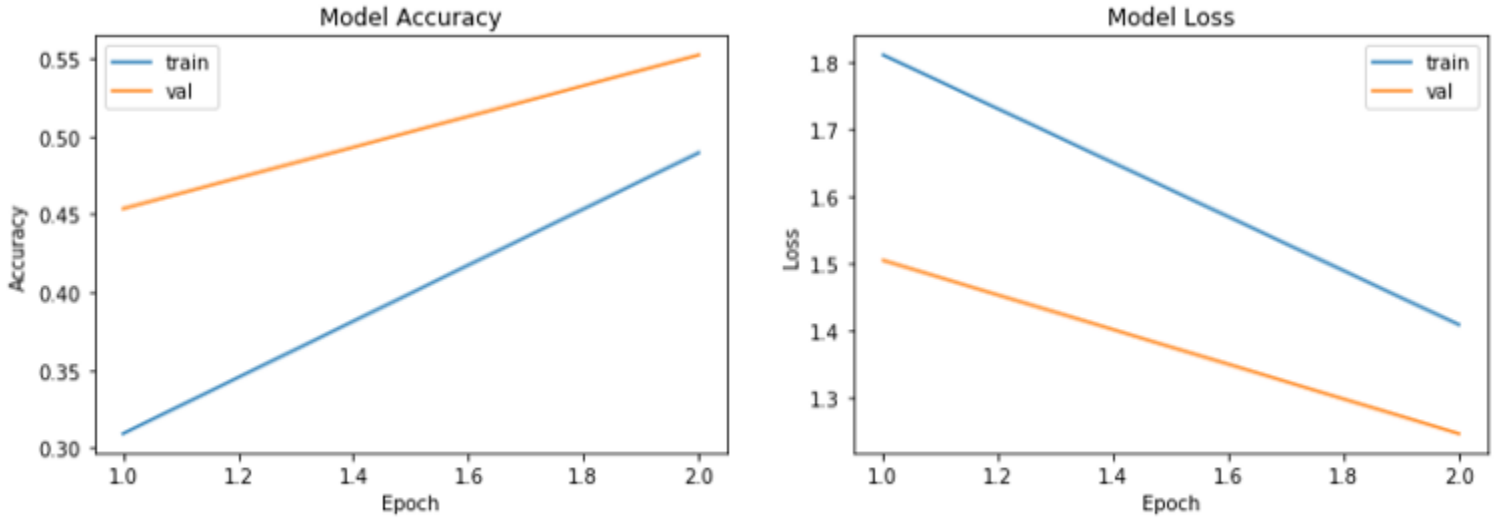
Table 1 – Our 1st Multilayer Model

We trained the previous model using the fit function and we iterated on our training data in 32 sized batches. Luckily, in Keras, we don't have to explicitly handle the batching up of our data during training, rather we just specify the batch size and it does it for us. In the beginning, we trained it for 2 epochs. From the results presented below, the model achieves a peak performance of about 55.52% accuracy, a pretty promising result which led us to the 2nd model as presented afterwards.

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/2
50000/50000 [==============================] - 513s - loss: 1.8110 - acc: 0.3093 - val_loss: 1.5051 - val_acc: 0.4536
Epoch 2/2
50000/50000 [==============================] - 505s - loss: 1.4093 - acc: 0.4895 - val_loss: 1.2472 - val_acc: 0.5522
```

Figure 14 – Our 1st Multilayer Model

Additionally, the metric curves follow an increasing trend line with the test dataset achieving better accuracy than the train one.




```
training time 1023.02 seconds
```

Figure 15 – Our 1st Multilayer Model, Accuracy and Loss

2ND MULTILAYER MODEL FOR 100 EPOCHS

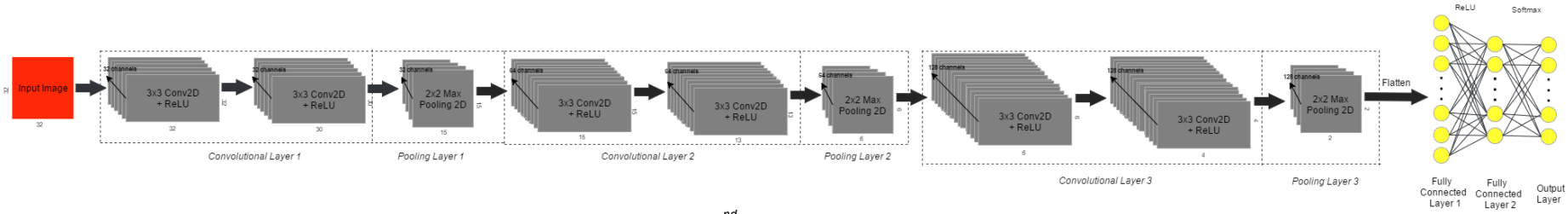


Figure 16 – Our 2nd Multilayer Model

The 2nd model consists of the same structure as the 1st one. The difference is that we trained it this time for <u>100 epochs</u> hoping to improve performance. The results of the 2 last epochs are shown below. It consists of 683562 learnable parameters and requires about 19.5M multiply-add operations to compute inference on a single image.

```
Epoch 99/100
50000/50000 [==============================] - 532s - loss: 0.7474 - acc: 0.7661 - val_loss: 0.7049 - val_acc: 0.7726
Epoch 100/100
50000/50000 [==============================] - 512s - loss: 0.7466 - acc: 0.7671 - val_loss: 0.6838 - val_acc: 0.7882
```

Figure 17 – Our 2nd Multilayer Model

This model achieves a peak performance of about 78.82% accuracy within a few hours of training time on a CPU. It is generally a highly applauded result. From the 30th epoch and after, the metric curves follow a steady trend as shown below.

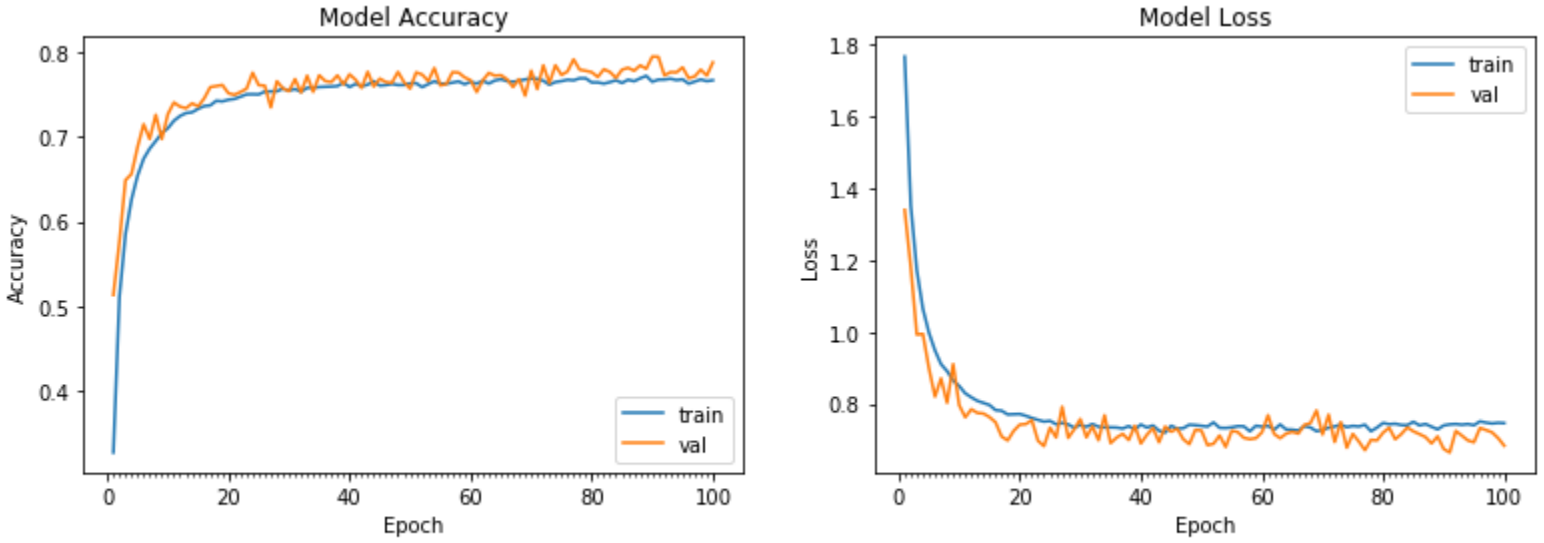


Figure 18 – Our 2nd Multilayer Model, Accuracy and Loss

Regarding the Confusion matrix given below, we observe that in general our model classifies more accurately the non-living targets of our images (airplane, ship, truck and auto). However, we can conclude that it gets more confused when it comes to classifying the living ones (bird, deer, cat, dog, frog, horse). More specifically, the dogs as being cats (185 mis-classifications). Weird enough, it confuses the birds as being deer (95 mis-classifications), as well as being cats (88 mis-classifications) or even airplanes (75 mis-classifications).

```
Confusion Matrix:
[836   5  14  36  10   1   3  10  60  25] (0) airplane
[ 13 867   1  14   1   1   9   1  25  68] (1) auto
[ 75   0 597  88  95  39  67  23   8   8] (2) bird
[ 27   0  26 659  49 133  63  20  13  10] (3) cat
[ 12   0  28  71 788  22  39  31   7   2] (4) deer
[  4   1  25 185  34 701  12  30   3   5] (5) dog
[  6   1  21  64  35  12 851   2   8   0] (6) frog
[ 12   1  15  63  43  50   3 806   4   3] (7) horse
[ 47  10   7  17   3   1   1   0 894  20] (8) ship
[ 19  35   2  24   3   0   5   3  26 883] (9) truck
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
```

Figure 19 – Our 2nd Multilayer Model, Confusion Matrix

Since this is the model with the best performance achieved, we deployed t-SNE visualization in order to get the topology of our classification image space by embedding them into 2 dimensions.

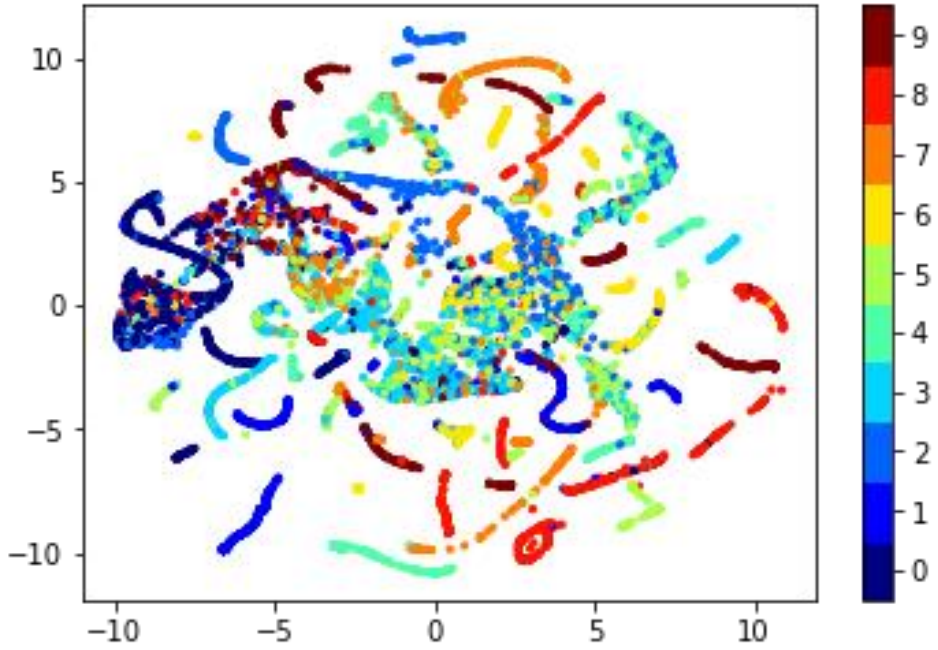


Figure 20 – t-SNE plot for CIFAR10

3RD MULTILAYER MODEL FOR 100 EPOCHS WITH DATA AUGMENTATION

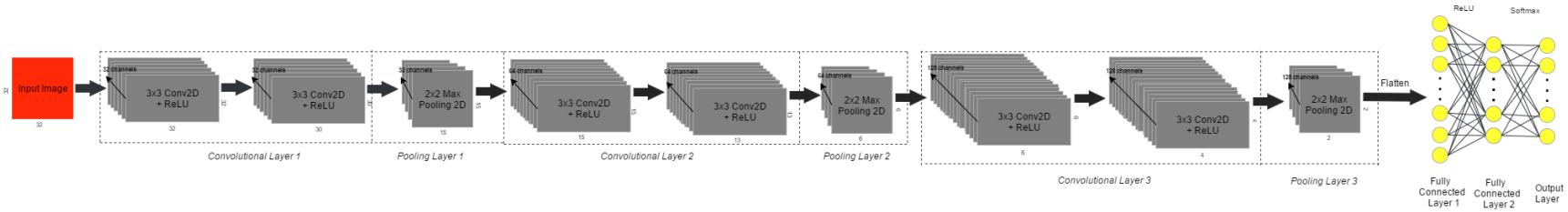


Figure 21 – Multilayer Model

In our deliverable, you will find the code referring to the same Multilayer model, mentioned before, in addition with data augmentation techniques. Due to the fact that data augmentation takes place again and again in every epoch separately, high computational resources and time are essentially required. Thus, we didn't manage to take results.

These data augmentation techniques have the following parameters:

- featurewise_center : set input mean to 0 over the dataset
- featurewise_std_normalization : divide inputs by std of the dataset
- vertical_flip : randomly flip images
- width_shift_range=0.2 : randomly shift images horizontally (fraction of total width)
- height_shift_range=0.2 : randomly shift images vertically (fraction of total height)
- rotation_range : randomly rotate images by 40 degrees
- zoom_range=0.2
- zca_whitening : apply ZCA whitening *
- fill_mode='nearest' : points outside the boundaries of the input are filled according to the nearest one

*ZCA whitening = Zero-phase Component Analysis which transforms the data locally, as little as possible, using Mahalanobis distance. ZCA whitened images still resemble original images, in contrast to PCA whitened ones. ZCA stretches the dataset to make it spherical, but tries not to rotate it, while PCA rotates it a lot.

4TH MULTILAYER MODEL FOR 50 EPOCHS WITH DATA AUGMENTATION

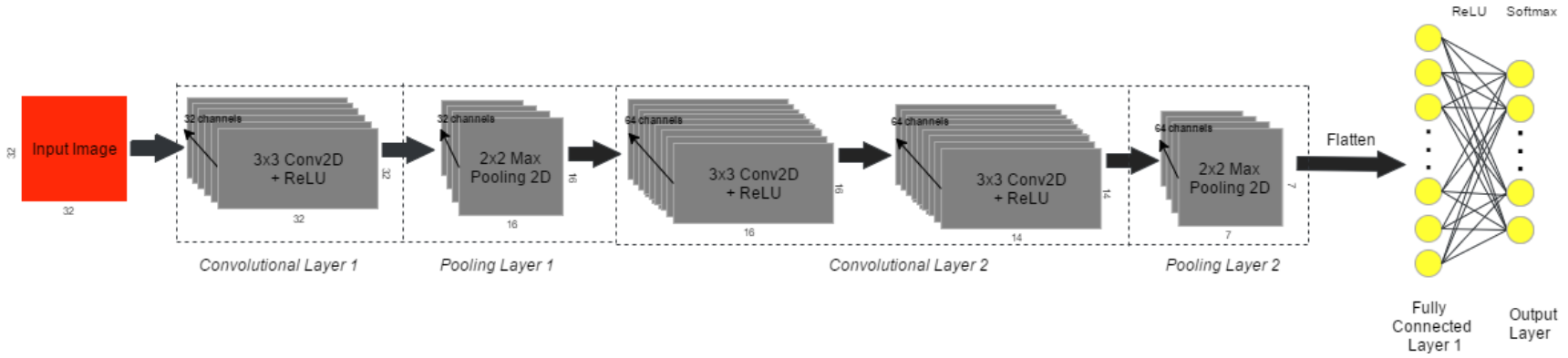


Figure 22 – Multilayer Model

This model follows a more simplified layer architecture than the previous ones. However, in order to combat overfitting, we deployed data augmentation with the following parameters:

- featurewise_center : set input mean to 0 over the dataset
- featurewise_std_normalization : divide inputs by std of the dataset
- vertical_flip : randomly flip images
- rotation_range : randomly rotate images by 25 degrees

The exact model structure is described in the following table.

Since the computational resources of our local machines were not sufficient to train on time such a huge model, we decided to use Microsoft Azure which dropped down the required time, from 10 days to 50 hours approximately. The model achieves approximately 77% accuracy.

Layer Name	Type	Filter size/ stride	Output size	Parameters
Conv2d_1	Convolutional	3x3 / 1	32x32x32	896
Activation_1	Activation		32x32x32	0
Max_pooling2d_1	Max Pooling		16x16x32	0
Conv2d_2	Convolutional	3x3 / 1	16x16x64	18496
Activation_2	Activation		16x16x64	
Conv2d_3	Convolutional		14x14x64	36928
Activation_3	Activation		14x14x64	
Max_pooling2d_2	Max Pooling		7x7x64	
Flatten_1	Flatten/Reshape		3136	
Dense_1	Fully Connected		512	3136 (input values) * 512 (neurons) + 512 (bias values) = 1606144
Activation_4	Activation		512	
Dropout_1	Dropout (50%)		512	
Logits/Cost (dense_2)	Softmax		10	[512 (input weights) +1 (bias weight)] * 10 classes = 5130
			Total/Trainable params:	1667594

Table 2 – Our 3rd Multilayer Model

The concept behind this experiment was to spot out which architecture performs better, in terms of accuracy and time: the one with the multiple convolutional layers or the one with less convolutional layers, but with augmented data.

PART V - CONCLUSION

The majority of visual recognition projects use pre-trained convolutional neural networks, such as Inception V3, ResNet, DenseNet, VGG-16 etc. based on the largest image dataset till today (ImageNet : 2 million images). This approach, called transfer learning, uses the architecture of the pre- trained CNN model and applies it on each dataset every time.

With transfer learning, we transmit the already stored knowledge and apply it on our problem of interest. This way we gain a lot of points regarding time and performance accuracy, by having already well-functioning parameters in our hands.

Last but not least, deep learning projects need financial recourses to perform successfully its computational procedures, in order to transform the data scientist into a decent human being.