**Home Work Assignment 1**

Or Kozlovsky 305447476

Noa Englender

**Task 1: Deep Learning Introduction**

**Dry section**

1. Working with a convolution network:
   1. the output dimensions of the following layers are with an Input RGB image of size 128X128X3:

|  |  |  |
| --- | --- | --- |
| Layer # | Layer description | output dimensions |
| 1 | convolution with 64 kernels of size 1X1X3 | 128X128X64 |
| 2 | max pooling of size 2x2 | 64X64X64 |
| 3 | convolution with 32 kernels of size 5X5X64 (no zero padding). | 60X60X64 |

* 1. We will explain the calculation of a 2D convolution with a kernel of size 1X1X3.

The output of the convolution is the sum of the same pixel in all 3 channels of the image. i.e. let’s assume we have the following kernel:



|  |  |  |
| --- | --- | --- |
| **1** | **1** | **1** |

And the following image channels:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** |  | **1** | **2** | **3** |  | **1** | **2** | **3** |
| **4** | **5** | **6** |  | **4** | **5** | **6** |  | **4** | **5** | **6** |
| **7** | **8** | **9** |  | **7** | **8** | **9** |  | **7** | **8** | **9** |

Then the output, as described above, will be:

|  |  |  |
| --- | --- | --- |
| **3** | **6** | **9** |
| **12** | **15** | **18** |
| **21** | **24** | **27** |

* 1. We will present the output of the convolution of the given sub-image with the following average filter:
     1. With stride = (1,1) and padding = (1,1):
     2. With stride = (2,1) and padding = (0,0):

1. We selcted to present the VGG16 classification architecture:

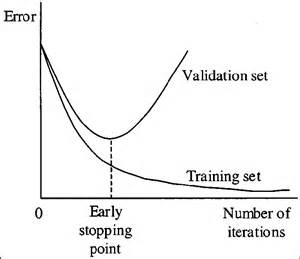
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Layer # | Layer description | output dimensions | Kernel size | Stride  (dx, dy) | Padding\dilation  (Dx, Dy) |
| 0 | input | 224x224x3 | - | - | - |
| 1 | Convolution | 224x224X64 | 3X3 | (1, 1) | (1, 1) |
| 2 | Convolution | 224x224X64 | 3X3 | (1, 1) | (1, 1) |
| 3 | max pooling | 112x112x64 | 2X2 | (2, 2) | (1, 1) |
| 4 | Convolution | 112x112x128 | 3X3 | (1, 1) | (1, 1) |
| 5 | Convolution | 112x112x128 | 3X3 | (1, 1) | (1, 1) |
| 6 | max pooling | 56X56x128 | 2X2 | (2, 2) | (1, 1) |
| 7 | Convolution | 56X56x256 | 3X3 | (1, 1) | (1, 1) |
| 8 | Convolution | 56X56x256 | 3X3 | (1, 1) | (1, 1) |
| 9 | Convolution | 56X56x256 | 3X3 | (1, 1) | (1, 1) |
| 10 | max pooling | 28X28x256 | 2X2 | (2, 2) | (1, 1) |
| 11 | Convolution | 28X28x512 | 3X3 | (1, 1) | (1, 1) |
| 12 | Convolution | 28X28x512 | 3X3 | (1, 1) | (1, 1) |
| 13 | Convolution | 28X28x512 | 3X3 | (1, 1) | (1, 1) |
| 14 | max pooling | 14X14x512 | 2X2 | (2, 2) | (1, 1) |
| 15 | Convolution | 14X14x512 | 3X3 | (1, 1) | (1, 1) |
| 16 | Convolution | 14X14x512 | 3X3 | (1, 1) | (1, 1) |
| 17 | Convolution | 14X14x512 | 3X3 | (1, 1) | (1, 1) |
| 18 | max pooling | 7X7x512 | 2X2 | (2, 2) | (1, 1) |
| 19 | Flatten | 1x25088 | - | - | - |
| 20 | Fully connected | 1x4096 | - | - | - |
| 21 | Fully connected | 1x4096 | - | - | - |
| 22 | Prediction (out) | 1x1000 | - | - | - |

* The output of one layer is the input of the sequential layer.
* The input size, kernel size, stride and padding dictamens the output size of each layer as describes in proviso sections

1. overfitting is when our trained model (net) doesn’t generalize well from our training data to unseen data. When a model learns the noise (training set) instead of the signal (observed (true) values) is considered “overfit” because it fits the training dataset but has poor fit with new datasets. We can recognize it for example, if our model saw 99% accuracy on the training set but only 55% accuracy on the validation set.

We can measure how well each iteration of the model performs When training a learning algorithm iteratively. Up until a certain number of iterations, new iterations improve the model. After that point, however, the model’s ability to generalize can weaken as it begins to overfit the training data.

Early stopping refers stopping the training process before the learner passes that point.



In the above graph we can recognize two cases:

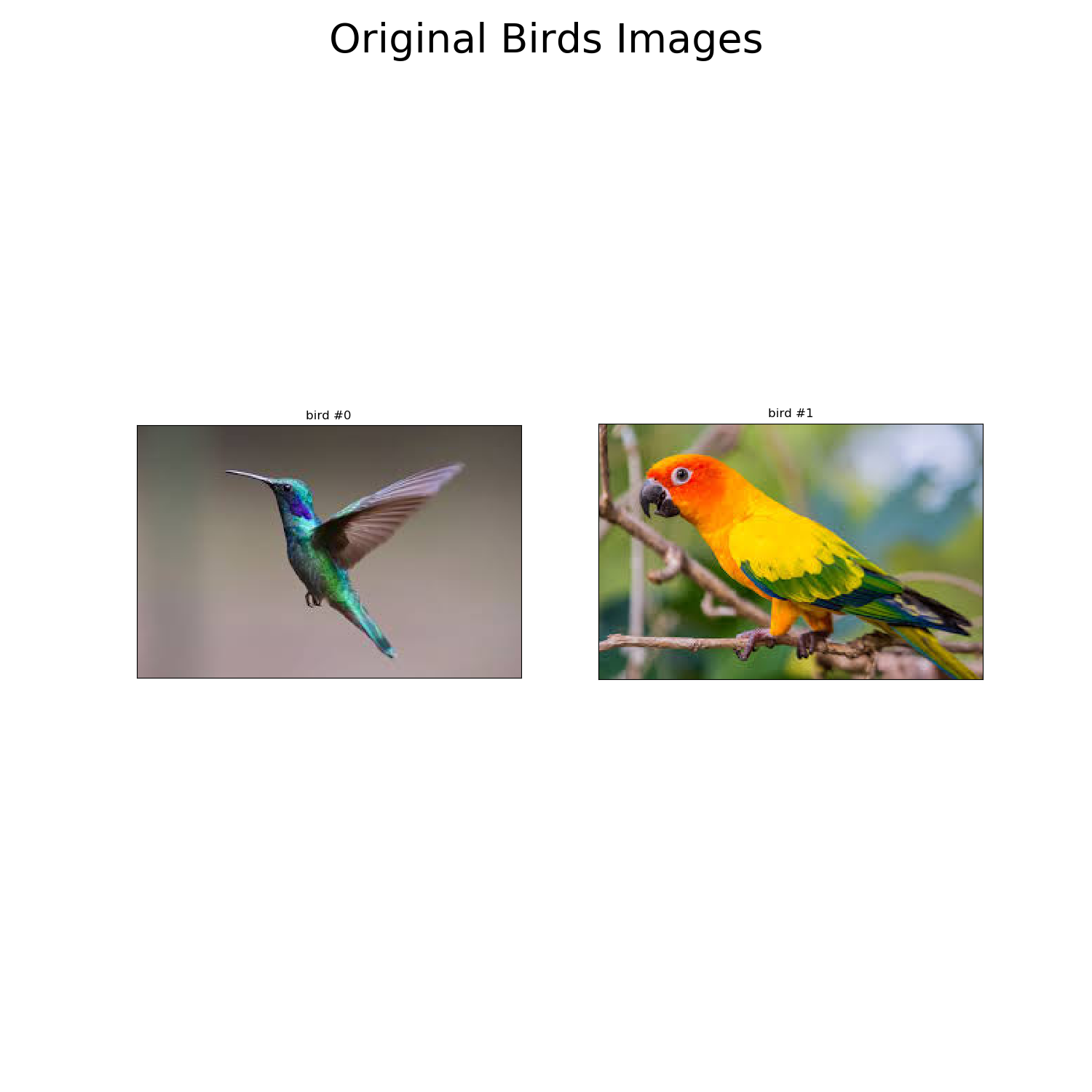
1. The error (accuracy) of the Training set which decrease as monotonously as the number of iterations.
2. The error of the Validation set which from a certain point begins to increase as the iterations continue.

Our goal when building our learning algorithm is to recognize this point and make sure we not passing it.

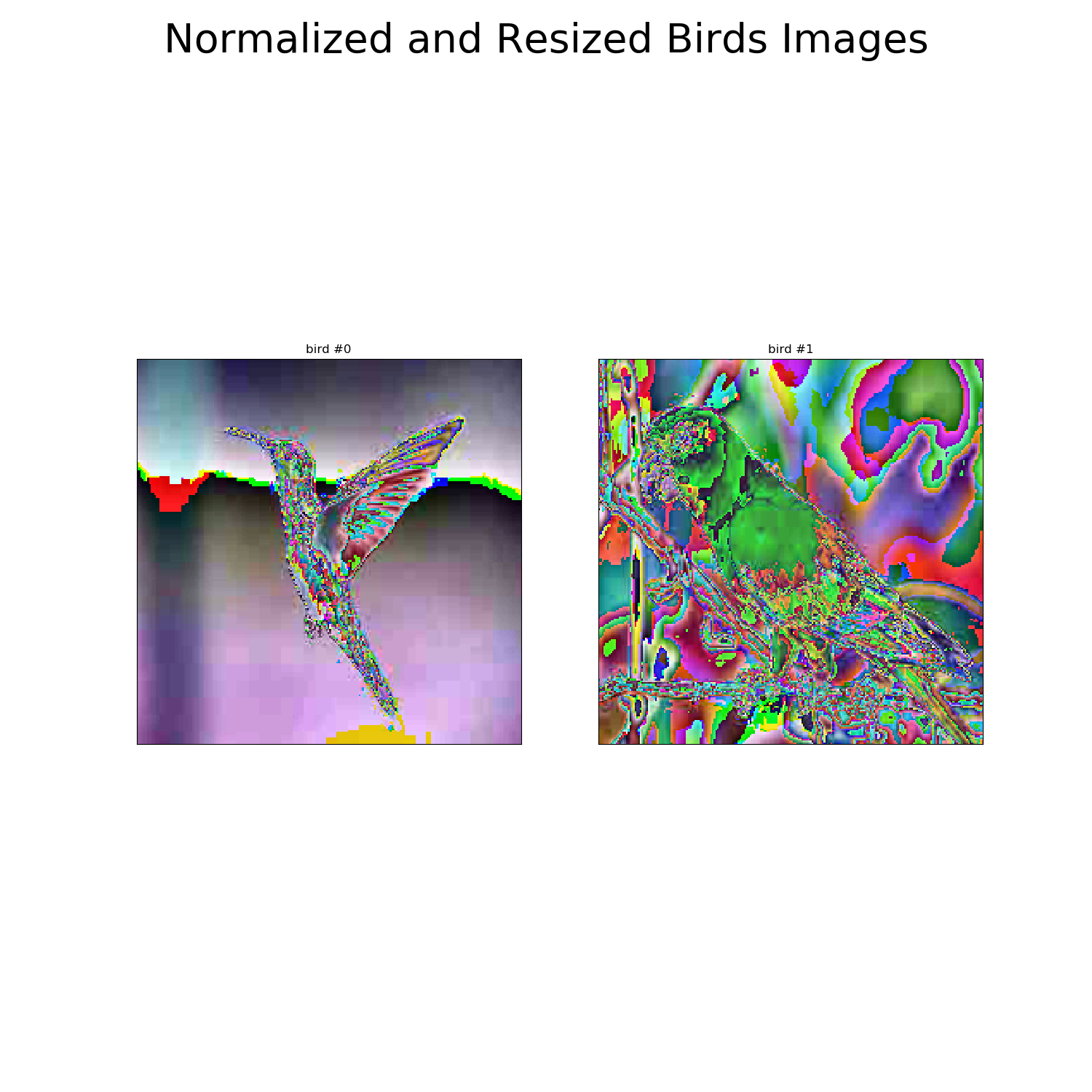
**Wet section**

1. Dsf

The Original birds image



The transformed image – fitting as an input to VGG16



The model succeed to classify the image:

**The model (VGG16) output**

|  |  |
| --- | --- |
|  |  |

1. On the lest is the original image we toke from the internet and on the right is the output of the model. The model succeed to classify it correctly as agama (a kind of lizard):

|  |  |
| --- | --- |
|  |  |

1. the resulting images of using one geometric transformation, one color transformation and one filter on the image above:



The corresponding outputs of the model for those image:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

**Task 2: Edge detection**

1. We will explain the way of operation of each of the three detectors:
   * Gaussian-Laplace: highlights regions of rapid intensity change and is therefore used for edge detection. It calculates the Laplacian of the image given by the relation, . the operator uses a 3x3 kernel which convolved with the original image to calculate approximations of the derivatives. to suppress the noise before using the Laplacian we use a Gaussian kernel.
   * Sobel: works like the first order derivate and calculates the difference of pixel intensities. The center column is of zero, so it does not include the original values of an image but rather it calculates the difference of right and left pixel values around that edge. Also, the center values of both the first and third column is 2 and -2 respectively.

This give more weight to the pixel values around the edge region. This increase the edge intensity and it became enhanced comparatively to the original image.

* + Canny: Canny edge detection is a multi-step algorithm that can detect edges with noise suppressed at the same time:
    1. Smooth the image with a Gaussian filter.
    2. Compute the gradient.
    3. Appling a threshold.
    4. Suppress non-maxima pixels to thin the edge ridges.
    5. Threshold the previous result by two different thresholds.
    6. Link edge segments in the two imaged from the proviso step.

1. We will explain the threshold parameter and the sigma parameter of the detectors:

|  |  |  |
| --- | --- | --- |
|  | threshold | sigma |
| Canny | Any edges with intensity gradient more than max threshold are sure to be edges and those below min threshold are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to “sure-edge” pixels, they are part of edges. Otherwise, they are also discarded. | The standard deviations of the Gaussian filter - determines the smoothing factor. For example, the sigma decreases then we give more weight to the pixels closer to the centered pixel and by that reducing the smutting effect. |
| Gaussian-Laplace | all values above the threshold determined as edges (1) and all below as non-edges (0) |
| Sobel | - |