Convolutional Neural Networks

Faur Anca Maria

May 4, 2018

Contents

1 Overview

This assignment aims to exemplify the use of the machine learning Tensorflow Framework in order to build a convolutional network classifier. The Keras high-level neural-network API used in order to enable fast experimentation.

The official websites proving information about tensorflow and keras are:

- `https://www.tensorflow.org/`
- `https://keras.io/`

1.1 Running instructions

1.1.1 Install instructions

1. I downloaded Python 3.6 version and introduced Python as enviromental variable

2. I downloaded the Anaconda installer and introduced Anaconda as enviromental variable

3. I created a tensorflow enviromental

   ```
   conda create -n tensorflow pip python=3.5
   ```

4. I activated the tensorflow enviromental

   ```
   C:> activate tensorflow
   ```

5. I followed the requirements for installing the GPU support from the official tensorflow website

   - CUDA Toolkit 9.0. .
   - The NVIDIA drivers associated with CUDA Toolkit 9.0.
   - cuDNN v7.0.

6. I installed the GPU tensorflow version using

   ```
   (tensorflow) pip3 install --upgrade tensorflow-gpu
   ```

1.1.2 Running example

For checking the installation, I run the following example:

1. I cloned a repository containing a tensorflow pre-made estimator

   ```
   git clone https://github.com/tensorflow/models
   ```

2. I tested the pre-made estimator on the Iris dataset

   ```
   cd models/samples/core/get_started/ python premade_estimator.py
   ```

3. I obtained the accuracy of predicting each class (Setosa, Versicolor, Virginica)

1.2 Introduction to Neural Network

1.2.1 Biological and Artificial Neuron

Our nervous system consists of about 100 billion interlinked neurons that are capable of carrying out complex computations. Each neuron has an antenna zone comprising the cell body and extremities, one axon and thousands of dendrites. Sufficient electrical activity on a neurons dendrites causes an electrical pulse to be sent down the axon, where it may activate other neurons.

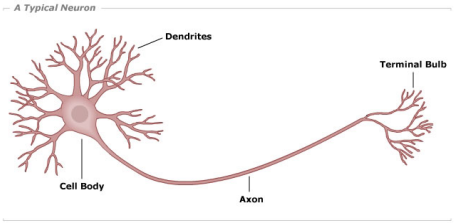

Figure 1: Biological neuron

An artificial neuron is a mathematical function conceived as a model of biological neurons and represents the elementary unit in an artificial neural network. The artificial neuron receives one or more inputs (as through the neural dendrites) and computes a weighted sum to produce an output (as the action potential which is transmitted along the axon).

$$\sum_{i \in [0,n]} = weight_i * x_i$$

The weighted sum is passed through an activation/transfer function.

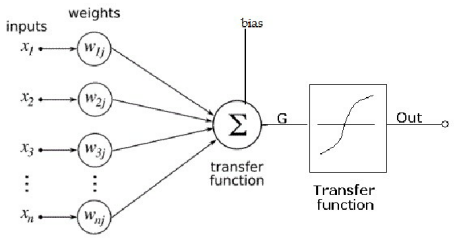

Figure 2: Artificial neuron

1.2.2 Activation Functions

An activation function is used in order to introduce non-linear properties into the network. It takes the weighted sum of the inputs, adds direction and decides whether to activate a particular artificial neuron or not.

- `Signoid function`

The sigmoid function or Logistic function is an activation function where it scales the values between 0 and 1 by applying a threshold.

$$f(x) = \frac{1}{1 + e^{-x}}$$

The figure below indicates the shape of the function.

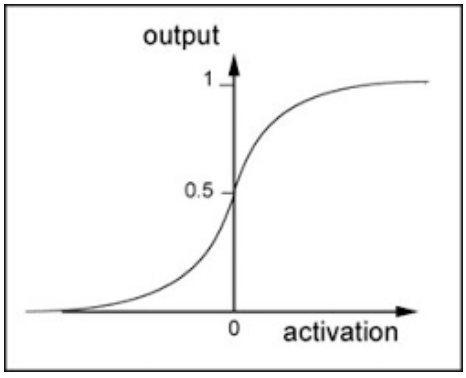

Figure 3: Sigmoid function

- `ReLU function`

  ReLU(Rectified Linear Unit) is one of the most widely used activation function. The benefits of ReLU is the sparsity, it allows only values which are positive and negative values are not passed which will speed up the process.

$$f(x) = max(0, x)$$

  The figure below indicates the shape of the function.

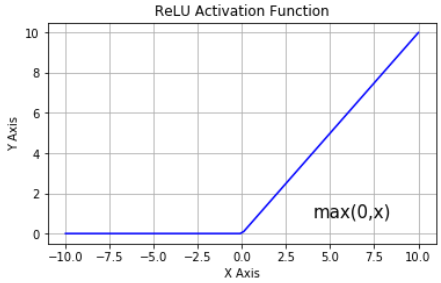

Figure 4: ReLU function

- `Softmax function`

  The softmax function is especially used when multi-class classification is desired. It squashes the outputs of each neuron to be between 0 and 1,

just like a sigmoid function, but it also divides each output such that the total sum of the outputs is equal to 1.

The output of the softmax function is equivalent to a categorical probability distribution, it tells you the probability that any of the classes are true.

$$f(x)_j = \frac{e^{z_j}}{\sum_{k=1..classes} e^{z_k}}$$

1.2.3 Neural Network Topology

Neural networks are composed of layers of neurons. They receive an input (a single vector) through the input layer, and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the output layer and in classification settings it represents the class scores.

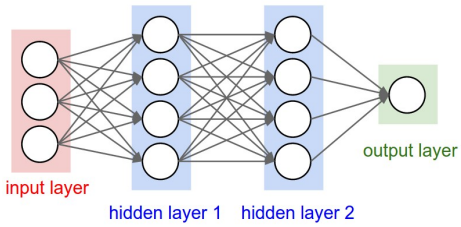

Figure 5: Neural Network Layers

The activation functions of the neurons depend upon their position in the layered structure.

- for the hidden layers, we might use the Sigmoid or ReLU activation functions
- for the output layer, if our model is a classifier, we would use Sigmoid activation function for binary classification and Softmax activation function for multi-class classification

Neural Networks don't scale well for images due to the high number of parameters (weights) that must be learned. For example, an image of more respectable size, e.g. 200x200x3, would lead to neurons that have 200*200*3 = 120,000 weights. Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

1.3 Backpropagation Algorithm

Backpropagation is a method used in artificial neural networks for the calculation of the weights for the inputs and connections between neurons. In the context of learning, backpropagation adjusts the weight of neurons by calculating the gradient of the loss function. This technique is also sometimes called

backward propagation of errors, because the error is calculated at the output and distributed back through the network layers.

Let's consider Mean-Squared-Error is used for the loss function in order to compute the error of the model.

$$J(X) = \frac{1}{m} * \sum_{i \in [1,m]} (y_i - yPredicted_i)^2$$

where
J = loss function
X = set of examples x_i i=1..m
m = no of examples in set X
y_i = the actual class of example x_i
$yPredicted_i$ = the predicted class of example x_i

1. `Random initialization:` initialize all weights randomly
2. `Forward propagation:` make predictions for all the training examples, layer by layer from layer 1 to layer L.
   - Calculate the inputs to the units in that layer (weighted sums)
   - Calculate the outputs of the units in that layer (using activation function)
3. `Backward propagation:` calculate the error signals at layer by layer in reverse, from layer L to layer 1.
   - Calculate the error signals for the units in that layer

     Consider the connection between a neuron of layer Lp connected to a neuron of layer Lp-1

     The error of the neuron of layer Lp is the multiplication between the activation function of the neuron of layer Lp-1 and the error signal of the neuron Lp-1 (formula for the error signal is too complicated for the purpose of this introduction)

     This relation shows that the errors on one layer are based on the error signals of subsequent layers.
   - Update all the weights according to the gradient of the loss of previous layer.

     The update is also influenced by a learning rate parameter which varies between 0 and 1. If this parameter is close to 0, the update of weights occurs slowly and more steps are required for the model to converge, but if the learning rate is too close to 1, the update is too drastic and the model may skip the optimal solution or even diverge.

     $$newWeight = oldWeight - \alpha * \frac{\partial J}{\partial w}$$

     where α = learning rate parameter

1.4 Introduction to Convolutional Network

Convolutional Neural Networks (convnets) are very similar to ordinary Neural Networks as both are made of layers of neurons that have learnable weights. However, they have a more sensible approach regarding the neurons connections. Convolutional Networks are widely used in computer vision and in other perceptual problems including speech recognition and natural language processing.

1.4.1 Hierarchical approach

Convolutional Networks model the primate vision system where there is a hierarchy of neurons within the visual cortex.

- In the lowest layers, neurons have small local receptive fields and respond to stimuli in a limited region of the visual field (e.g. spots of light)
- In higher layers, they combine the outputs of neurons in the lower layers and are able to respond to more complex stimuli (e.g. lines at particular orientations)
- In the highest layers, they respond to ever more complex combinations, such as shapes and objects

As a result, the convolutional networks are able to learn features that are translation invariant (e.g. the cat can be recognized no matter its position in the image)

1.4.2 Layer Types

- Dense Layers

  Dense layers or fully connected layers connect every neuron in one layer to every neuron in another layer.

- Convolutional Layers

  A convolutional layer is a 3D tensor of neurons, whose shape is (height, width, depth). The depth represents the number of feature maps, each feature map in a layer is responsible for learning a certain feature. Within one feature map, all neurons share the same weights resulting in the translation invariance of the learned features.

  In a convolutional layer, every neuron in that layer has connections from only a small rectangular window of neurons in the preceding layer, typically 3x3 or 5x5.

- Pooling layers

  Pooling layers are used to shrinks the number of neurons in higher layers in order to:

  – reduce the amount of computation

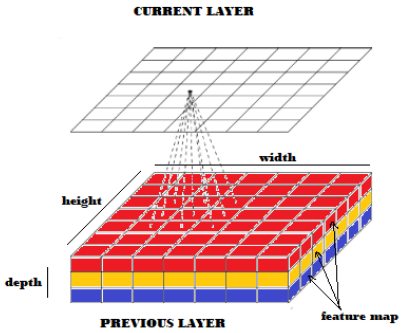

Figure 6: Convolutional Layer

- reduce the number of parameters to be learned
- create a hierarchy in which higher convolutional layers contain information about the totality of the original input features

The pooling layers work on rectangular windows: neurons in the pooling layer are connected to windows of neurons in the previous layer.

MaxPooling is a particular type of pooling layer where a neuron receives the outputs of the neurons in the window in the previous layer and outputs only the largest of them.

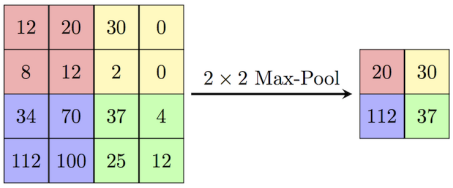

Figure 7: Max Pooling Layer

1.4.3 Convolutional Network Topology

A usual convolutional network is composed of several convolutional layers separated by pooling layers, followed by a flatten layer that transforms the tensor of neurons in a 1D array and the last layers are fully connected layers.

The figure below shows an example of a convolutional network that is used to classify hand-written digits. The inputs of the network are 28x28 grey-scale images and there are 10 output neurons, each one activating for a digit between 0-9.

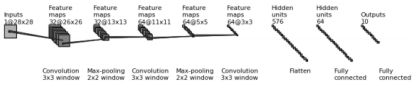

Figure 8: example of Convolutional Network

1.5 The overfitting problem

1.5.1 Detecting overfitting

Overfitting occurs when we are training a model that is too complex for the amount of the trainig data and for its noisiness. As a result, the learning algorithm fits too well the examples of the train set (including its outliers), but loses its power of generalization. When it is supposed to analyse new examples, e.g. from the validation/test sets, we notice that the loss function outputs an error that is way above the error on the train set.

Overfitting can be easily spotted if we plot the error of the training and validation set. In the case of overfitting, the training error is small and there remains a big gap between training error and validation error.

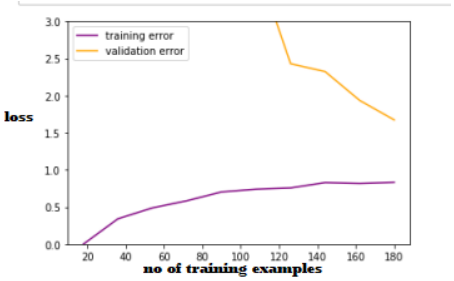

Figure 9: example of overfitting

The figure also analyses how the number of training examples influences the learning, we notice that, until a certain point, the increase of the number of examples improves the quality of the model.

A model that is suitable for its training examples would have a plot as the one below. We can see that as the number of examples learned increases, the error decreases. Moreover, the gap between the train error and the validation error narrows and they converge.

1.5.2 Solutions for Neural Networks

1. Reducing the network's size

   A neural network can be simplified by reducing the number of parameters, through reducing the number of hidden layers and/or the number of neurons inside the hidden layers.

2. Weight regularization

Figure 10: example of a suitable model

The complexity of the model can be reduced by reducing the range for the values of the weights parameters. Weight regularization means forcing the weights values to be small through evaluating the model using a loss function that is conditioned by the small values of the weight parameters also.

- Lasso: we penalized by the l1-norm (the sum of their absolute values of the weights)
- Ridge: we penalized by the l2-norm (the sum of the squares of the weights)
- A hyperparameter , called the 'regularization parameter' controlled the balance between fitting the data versus shrinking the weights

3. Dropout

   The complexity of the model can be reduced by altering the number of neurons during training. We give neurons a probability `p` to be dropped out in the training process.

   For every mini-batch in the training set:

   - In the forward propagation, decide which neurons will be dropped with probability `p`. Set the activations of the dropped neurons to zero and divide the activations of the kept neurons by `(1-p)` (as the kept neurons will receive inputs from more neurons)
   - In the backpropagation, ignore the dropped out neurons

2 Proposed problem

2.1 Specification

This assignment aims to build a classifier able to recognize between 10 different species of monkeys. The classifier is built using the Convolutional Neural Network technique. For machine learning, the Tensorflow framework is used with the help of the Keras high-level neural network API.

2.2 Data representation

The dataset used consists of JPEG format images of 10 different species of mokeys. The data is divided into 2 folders, training and validation. The folders contain 10 sub-folders labelled as n0 n9, each containing images of a different species of monkey. The total number of images is around 1400. The dataset can be found at the link:

`https://www.kaggle.com/slothkong/10-monkey-species`

The images are decoded and uncompressed in grids of 150x150 RGB pixels. The values of the pixels would be too high for the model to process (given a typical learning rate), so values are scaled between 0 and 1 using 1/255 factor.

2.3 Implementation Overview

The problem of classifying the monkey species is solved using convolutional networks. The base network is composed of 4 convolutional layers (and max-pooling layers in between) and 2 fully connected layer. In order to respond to the problem of overfitting, the base network is then slightly modified resulting in the network having weight regularization and the network using dropout. The number of examples is artificially increased using data augumentation.

Moreover, the assignment exemplifies the use of a pre-trained model in order to solve the monkey problem.

The proposed neural networks have an accuracy between 60% to 70%.

2.4 The Base Convolutional Network

The following Keras segment of code presents the base convolutional network.

Figure 11: base network

Each convolutional layer has the following parameters (in order):

- the number of feature maps
- the size of the convolutional window between 2 consecutive layers
- the activation function

The last two layers are fully connected (dense), the first parameter represents the number of neurons and the second one represents the activation function. In order to classify between 10 classes, the last layer contains 10 neurons and softmax function is used in order to obtain a probability distribution between the classes. The class predicted with the highest probability is the predicted class.

In order to optimize the predictions, I have chosen the RMSprop variant of gradient descent for backpropagation calculus. The learning rate has been set to 0.0003.

The structure of the network can be better understood from the following summary.

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_9 (MaxPooling2	(None, 74, 74, 32)	0
conv2d_10 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_10 (MaxPooling	(None, 36, 36, 64)	0
conv2d_11 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_11 (MaxPooling	(None, 17, 17, 128)	0
conv2d_12 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_12 (MaxPooling	(None, 7, 7, 128)	0
flatten_3 (Flatten)	(None, 6272)	0
dropout_5 (Dropout)	(None, 6272)	0
dense_5 (Dense)	(None, 512)	3211776
dense_6 (Dense)	(None, 10)	5130

Total params: 3,457,738
Trainable params: 3,457,738

Figure 12: Network Summary

The learning process hyperparameters have been set to:

- number of epochs = 30 (epoch = pass over the all training examples)
- number of steps_per_epoch = 57 (in order to cover all 1097 training examples in 20 epochs)
- number of validation_steps=13 (in order to cover all 272 validation examples in 20 epochs)
- the batch size = 20 (the batch size = no of examples propagated through the network at once. A smaller number saves memory and improves speed due to simpler backpropagation updates on weights. However, the updates of the weights can be misleading as they are influenced by the prediction over a small partition of the dataset, not on the entire dataset).
- early_stopping = 2 (in order to speed up the learning process, the model stops training when the loss function has stopped improving for more than 2 epochs)

The initial prediction has an accuracy of approximately 69%. The following image shows the evolution of the loss function and accuracy metric during the first epochs until the process has been interrupted due to early stopping.

In the last epoch, we notice that there still is a big gap between the training and the validation examples. That gap suggests the problem of overfitting.

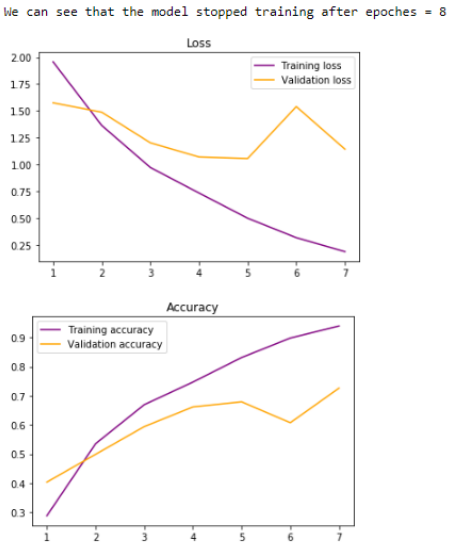

Figure 13: Loss and Accuracy evolution in epochs

2.5 Strategies against Overfitting

2.5.1 Data Augmentation

Data augmentation is another way we can reduce overfitting on models, where we increase the amount of training data using information only in our training data. However the obtained examples are not as good as new examples.

In order to obtain more examples, I used image transformations on the training images.

```
augmented_train_data_generator = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,                  # degree range for random rotations
    width_shift_range=0.2,              # range for random horizontal shifts
    height_shift_range=0.2,             # range for random vertical shifts
    zoom_range=0.2,                     # range for random zoom
    horizontal_flip=True,               # randomly, whether to flip inputs horizontally
    fill_mode="nearest")                #  the strategy for filling-in newly created pixels that appear
                                        # after some of the other transformations
```

Figure 14: Image transformations used for data augmentation

2.5.2 Weight Regulation

For weight regularization, the convolutional network uses Ridge l2-norm.

2.5.3 Dropout

Dropout is usually used in the dense layers as the number of parameters of the convolutional layers isn't considered big enough to result in overfitting.

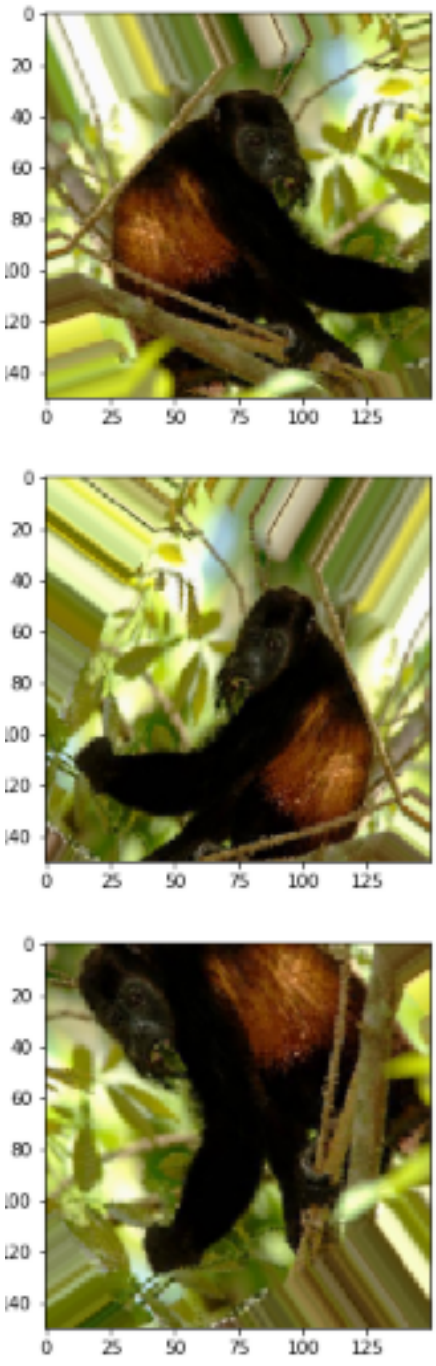

Figure 15: Images obtained due to data augmentation

```
def build_regulated_convnet():
    network = Sequential()
    network.add(Conv2D(32, (3, 3), activation="relu", input_shape=(150, 150, 3)))
    network.add(MaxPooling2D((2, 2)))
    network.add(Conv2D(64, (3, 3), activation="relu", kernel_regularizer=regularizers.l2(0.01)))
    network.add(MaxPooling2D((2, 2)))
    network.add(Conv2D(128, (3, 3), activation="relu", kernel_regularizer=regularizers.l2(0.01)))
    network.add(MaxPooling2D((2, 2)))
    network.add(Conv2D(128, (3, 3), activation="relu", kernel_regularizer=regularizers.l2(0.01)))
    network.add(MaxPooling2D((2, 2)))
    network.add(Flatten())
    network.add(Dense(512, activation="relu", kernel_regularizer=regularizers.l2(0.01)))
    network.add(Dense(10, activation="softmax", kernel_regularizer=regularizers.l2(0.01)))
    network.compile(optimizer=RMSprop(lr=0.0003), loss="categorical_crossentropy", metrics=["accuracy"])
    return network
```

Figure 16: Convolutional Network with weight regularization

```
def build_dropout_convnet():
    network = Sequential()
    network.add(Conv2D(32, (3, 3), activation="relu", input_shape=(150,150, 3)))
    network.add(MaxPooling2D((2, 2)))
    network.add(Conv2D(64, (3, 3), activation="relu"))
    network.add(MaxPooling2D((2, 2)))
    network.add(Conv2D(128, (3, 3), activation="relu"))
    network.add(MaxPooling2D((2, 2)))
    network.add(Conv2D(128, (3, 3), activation="relu"))
    network.add(MaxPooling2D((2, 2)))
    network.add(Flatten())
    network.add(Dropout(0.5))
    network.add(Dense(512, activation="relu"))
    network.add(Dense(10, activation="softmax", kernel_regularizer=regularizers.l2(0.01)))
    network.compile(optimizer=RMSprop(lr=0.0003), loss="categorical_crossentropy", metrics=["accuracy"])
    return network

```

Figure 17: Convolutional Network with dropout

Even if the classification accuracy doesn't improve, from the following images we can notice that the gap between the training examples and the validation examples loss and accuracy was reduced.

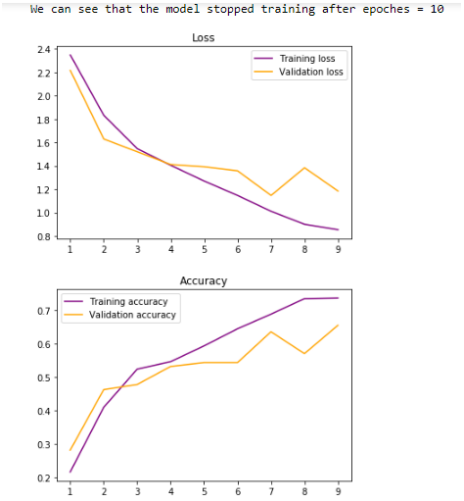

Figure 18: Loss and accuracy evolution in epochs when using dropout

2.6 Transfer Learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

Image classifying networks, especially if they contain few examples in the training set, can use pre-trained models. The pre-trained models have been trained on large datasets such as the ImageNet 1000-classes images (which recognize 1000 different objects).

The use of pre-trained models makes sense due to the hierarchical approach of convolutional networks. If the trained model is very good at recognizing

animals we can use its layers in order to detect features that interest us in the monkey classification problem as fur type, eye shape and so on. The trick consists in modifying the last layer of the pre-trained model in order to match our classification problem (to have 10 neurons,one for each monkey species). The update of the weights is done only on the connection between the last layer of the pre-trained model and the added last layer (fully connected layers).

Moreover, for obtaining great results, it is important that the pre-trained model had used images similar to the images of the new classifier. For example, a pre-trained model on medical tomography images would result in a low-performance classifier for images from outer space.

I have used the ResNet50 pre-trained model and the resulting classification accuracy is around 60%.

A furthermore improvement would be fine-tuning the model which means allowing the weight updates also on the last layers of the pre-trained model. The first layers can be "frozen" as they are good at detecting the simple shapes.

3 Conclusion

This assignment aimed to classify 10 species of monkeys. The accuracy of the obtained model is between 60% and 70%, I think this a good score considering the similarities of the monkeys and the reduced size of the training dataset. Moreover, this assignment gives an introduction to deep learning and explains different strategies for improving image classification.

4 Biography

1. biological neuron

   `http://neurosciencenews.com/neurons-synapses-neuroscience-5119/`
   `https://online.science.psu.edu/bisc004_activewd001/node/1907`

2. artificial neuron

   `https://en.wikipedia.org/wiki/Artificial_neuron`

3. activation functions

   `https://analyticsindiamag.com/most-common-activation-functions-in-neural-networks-and`
   `https://github.com/Kulbear/deep-learning-nano-foundation/wiki/`
   `ReLU-and-Softmax-Activation-Functions`

4. topology of layers

   `https://www.pyimagesearch.com/2016/09/26/a-simple-neural-network-with-python-and-kera`

5. neural network

   `http://www.cs.ucc.ie/~dgb/courses/ai2/08_neural_networks.pdf`

6. convolutional network

   `https://en.wikipedia.org/wiki/Convolutional_neural_network` `http:`
   `//www.cs.ucc.ie/~dgb/courses/ai2/13_convnets.pdf`

7. conv2D

   https://www.tensorflow.org/api_docs/python/tf/nn/conv2d

8. backpropagation

   http://www.cs.ucc.ie/~dgb/courses/ai2/10_backprop.pdf

9. overfitting

   http://www.cs.ucc.ie/~dgb/courses/ai2/05_over_n_under_fitting.
   pdf http://www.cs.ucc.ie/~dgb/courses/ai2/12_nn_overfitting.pdf
   https://chatbotslife.com/regularization-in-deep-learning-f649a45d6e0
   https://blog.keras.io/building-powerful-image-classification-models-using-very-little
   html

10. transfer learning

    https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-ke
    https://flyyufelix.github.io/2016/10/08/fine-tuning-in-keras-part2.
    html

11. data augmentation

    http://cs231n.stanford.edu/reports/2017/pdfs/300.pdf