Sprechtext

1. Welcome – Present Project
2. Our project was to create an AI which can convert old / grayscaled photos to colorful photos. Our main idea was to use the keras framework because it has so much power in setting up an artificial intelligence projects. The next step was to collect data. In our first version we only started with a few photos which we have downloaded from a kaggle dataset. We thought a size of 256x256 of the photos is already high enough because of performance and also the fact that we both do not have access to some GPU so we trained the most time only with CPU. In our case we took 256x256 sized pictures which should be colored. So the properties are: colored photos, 256x256 size, different categories. In our first set we only took about 8 pictures to prove that our approach is working. As next we downloaded a dataset with 256 categories with different sized photos. So we wrote a phyton script to reshape the images to our predefined size. At the end we manually picked out photos which are stretched to much and doesn’t look good.
3. We took RGB valued pictures which have color values between [0-255]. As next we planned to setup a sequential model and use convolutional layers to build our final model with some trainable parameters. Further we have trained the model and in the end we did the converting part.
4. This black and white image can be represented in grids of pixels. As is said the values are between 0-255 from black to white. Colored images have three layers Red, Green, Blue for each color.
5. So our idea was to link a grid of grayscale values to the three color grids. To ease this up we converted the RGB Spectrum to LAB spectrum because here we have one layer for grayscale and only two layers for color. So we can use the original grayscale image in our final prediction and we also only have to predict two color channels and not three.
6. Now on this slide you see all of our imports. As said we use the keras framework with different layers like Conv2D, UpSampling and so on. We also import the Sequential models part. To perform some image manipulation strategies we also imported skimage with different features like rgb2lab color as you will see. Then of course numpy and the matplotlib for drawing some fancy diagrams.
7. On this slide you can see how we stored our data in an array through running a loop. We then convert the data array to a float array of images with the np.array command. As last we set up our train data Xtrain. Here we have to consider RGB values which are in range of 0-255 so we have to divide it to produce values between 0 and 1.
8. Now come the funny part, building our model. And we also have to clarify which models from the literature are suitable for this problem given your data?

In our case we thought about a sequential model from keras which can be build up as a deep learning approach where the output from a previous neural layer serves as the input to the next layer. Why not a single layer neural network? Because single layer models can only be used to represent linearly separable function. This means very simple problems but in our case we thought we need a multiple layer neural network with some hidden layers. Here the question arises, how many layers should we use? Unfortunately there is not rule of thumb to calculate this so we had to experiment with trial and error, but we will see that soon.

In Keras there were also functional layers which allows you to connect a layer to any other number of layers in the network. Functional layer provide more flexibility to develop a very complex network with multiple input or outputs and the opportunity to share layers, but in our case sequential is sufficient.

1. Then there was the question - **which ML algorithms can be used to train them?**

There are many of machine learning algorithms out there but as we learned during the course we can divide it in four categories. Supervised, Unsupervised, Semi supervised and Reinforcement learning. There are also the keywords features and labels.

Features = properties of training data , colored values of photos / bw photos

Labels = output after training, we want to predict colors

Supervised = set of labeled training data, algo learns to predict the output from input data – decision trees

Unsupervised = Only input data – no labeled data, algo learns the inherent structure from the input data. group customers .. k means (divide data in k clusters)

Semi supervised = partially labeled input data, (too much time to label all images – photo classifier), mix of supervised and unsupervised techniques

Reinforcement learning = no training data, create agent which performs actions, learns through rewards from the environment, human way to learn .. (alpha GO)

We used semi supervised learning were we get sample data with concrete goal variables on which will trained as well as anonymous data on which will be tested. Advantage: we can train with only a few photos and make a prediction for the test images. This is done by creating an artificial neural network. In our case it is a CNN a convolutional neural network.

1. Convolutional layers = The convolutional neural network, or CNN for short, is a specialized type of neural network model designed for working with two-dimensional image data. In the context of a convolutional neural network, a convolution is a linear operation that involves the multiplication of a set of weights, the multiplication is performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel. The two-dimensional output array from this operation is called a “feature map“. Once a feature map is created, we can pass each value in the feature map through a nonlinearity, such as a ReLU, but we will see that soon. Here you see also a picture on how we would like to set up our CNN. Basically we see here that we will use different layers with different filters or neurons, which grows and shrinks and the end we build up our initial size of the image, the a und b color channels and as well as adding grayscale to finalize Lab image.
2. As you will see we will also working with Upsampling2D layer = double each row and column.

Strides (2,2) = move kernel - values are related to the two dimensions x and y, They will be incremented one by one, so first you will go along the x axis 2 by 2, without changing y, the you will go down y by one step of 2 and start over along x, and so on although

1. Now here we build up our neural network by using sequential models from keras – which is a deep learning approach. As first we create our model with the command Sequential(). The First time we add a model we have to define the input\_shape which is in our case 256x256 and 1 for 1 color space. This model add function is pretty simple. It adds new layers to our model and we can specify how many neurons or filters we want, we can specify the kernel space and we also have this activation functions like sigmolt or relu (in our case we need relu), and of course a padding which decreases the kernel space by filling up cells with zeros. We also use strides which represents a nr of tuples or how much kernel should be moved in our case 2 px. With that in mind we build up our model from low filters to high filters (512) in an order of 2 (power of 2) and back to low filters. In between we use the UpSampling2D layer to blow up or double our model vector. The final activation has to be a Tanh activation function because otherwise we would only get positive numbers a no negative numbers, but the blue and green spectrum of LAB is with negative numbers. At the end we call the compile function which takes some parameters like the optimizer, the loss and if you want the metrics. We tried with different optimizer like adam and RMSProp or Root mean square propagation which divides the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight. We then call model.summary() to get an overview about the produces layers, the output as well as the size of trainable parameters. We tried different models, especially in the beginning models with only 50k trainable parameters but it turned out that this is not sufficient so we blowed the model up till we thought ok that’s a good size of the parameters.
2. Setup ImageDataGenerator - How does it work?

1) accepting batch of images

2) apply random transformation to each image

3) replace original batch we newly transformed batch

4) train CNN on newly transformed batch

define function to call ImagedataGenerator

Setup Training

keras support 3 types of training

1.fit, 2. fit\_generator, 3. train\_on\_batch

* to train on large models **.fit** would not be suitable because RAM overfilled
* **.fit\_generator** - here data is no longer “static” — the data is constantly changing
* Each new batch of data is randomly adjusted according to the parameters supplied to ImageDataGenerator .fit\_generator accepts the batch of data, performs backpropagation, and updates the weights in our model
* **.train\_on\_batch** -- fine-grained control over training your deep learning models - mostly not suitable

1. History Overview
2. Save & Load models/weights
3. Colorizer section

Collect all test images into an array

Covert the color space and reshapre image

Call predict function

Print output which must be multiplied with 128 because of LAB color spectrum

Loop and make a 256x256 vector with 3 layers, copy the first grayscale layer to new object, copy the two color layers to new object and save the result in directory

?? auf die fragen noch bissi besser eingehen, absprechen..?