Intrinsic Image Decomposition

Optional Semester Project

Leonardo Aoun[1] supervised by Jan Bednarik[2]

***Abstract*— This report explains the work done during the Fall Semester 2017/2018 for the Optional Project. Done at the Computer Vision Lab, we decided to tackle the problem of intrinsic image decomposition, specifically: from an image I, get the shading and reflectance components. We did that by implementing a few Deep Neural Networks models and test their results on our generated dataset. With great results on our synthesized data, it is possible that with more training or fine tuning on more general, publicly available dataset, we could witness a good tool to use in problems such as replacing a texture in an image while preserving the shading, or estimating depth of objects.**

I. INTRODUCTION

Intrinsic Image Decomposition is a hard Computer Vision task which consists of decomposing an image into different images that when operated together, lead back to the same image. This is not just a mere information split or matrix factorization but the resulting images represent useful information about the original image, usually the shading and albedo components, all linked by the relation: $I = S * A$, where * is the pixel-wise multiplication. Some research studies even did it with specular and diffuse components or even shape and depth but we limited the scope of this project to the first two. The shading component actually describes the effect of lighting in the image while albedo (or reflectance) means the actual true color at an object. Due to the scarcity of real world images accompanied by their decomposed intrinsic components we focused on synthetic data and I used the Blender generated data by my supervisor. We decided to give Deep Learning a shot because it has been seeing the best results in recent papers related to this problem, shallow approaches like in [6] require more input data to be as successful, like face priors in this case.

II. PREVIOUS WORK

A. DARN [5]

This paper seemed to be the closest to what we were looking for. They presented a deep neural network resembling ResNet to extract the shading S and albedo A components from the image I. They follow the relation $I = \alpha A * \frac{S}{\alpha}$. So they first predict S from I and then divide the original input by S and feed A and S to a Generative Adversarial Network. The loss function they use is a combination of MSE for both the S and A components, as long as another MSE for the gradients of these components. The dataset used is the MPI Sintel dataset, also a synthetic dataset similar to our plan.

[1]Computer Science Master's student at EPFL, Switzerland
[2]PhD Student at the Computer Vision Lab of EPFL

B. Direct Intrinsics [7]

Researchers from Berkeley, TTI and Sony preferred to use a CNN more similar to a Fully Convolutional Network. They also showcased their work on the MPI Sintel Dataset. Their model was inspired by the multi-scale model of Eigen and Fergus. The first one is a coarse subnetwork while the second is a finer scaled one. Unlike [5], their model outputs the two predictions directly without dividing I by S. This means that the energy is not necessarily conserved.

C. Revisiting Deep Intrinsic Image Decompositions [4]

This paper" authors came up with an interesting structure for their model. They combined three main parts: The first being a direct intrinsic network to predict a preliminary reflectance image. In parallel, they pass the image and its input edge map to a guidance network to get a guidance map. Combining it with the predicted preliminary reflectance as an input to a domain filter network which generates a realistically flattened albedo image.

III. DATASET

A. Publicly available datasets

We had the option to choose from 3 different datasets to undergo our training.

- (a) The MIT Intrinsic dataset which consists of photographs taken of real objects. The original image is initially taken, then it is coated with a special material to remove specularity and more lights are added to the scene in order to capture its reflectance. Then a replica of the object, but all white replaces it so the shading with respect to the original lighting can be captured.
- (b) The MPI Sintel dataset [3] is a collection of 18 scenes from the Sintel short movie originally intended to evaluate optical flow techniques. Around 50 images are provided for each scene, along with their albedo components and optical flow groundtruth.
- (c) Intrinsic Images in the Wild (IIW) [2] is a crowdsourcing pipeline where people are asked to annotate features through an AWS platform. Basically each vote of each user consists of selecting which of the two pixels presented to them has a darker reflectance value.

Unfortunately, we ran into issues with each of these options. (a) was very small, it only contained 20 objects with 11 different lighting possibilities. The shading image they provided only corresponded, and not completely, to one of them, and the others could not easily be calculated: Dividing the provided image with the albedo was creating overflows due to very low values in some pixels of the object,

what most of the papers did was to calculate S following $I' = \alpha S * \frac{A'}{\alpha}$ where I' and A' are the averages over the 3 channels and α is the float that minimizes the loss of $I - [S] * [A]$ where [S] and [A] are S and A clamped between 0 and 1. This process of minimizing the loss of information due to clipping after trying to get the ground truth shading component would have also been a challenge had we decided to work with the MPI Sintel Dataset since they didn't provide any shading ground truth. What [7] and [5] did was to calculate this optimal shading, then resynthesize the image by multiplying this shading with the provided albedo. In addition, the dataset had a couple of flaws that could have made our model less applicable in real life scenes like the brownish vibe and the fluorescent lightings. IIW needed a lot of checking and validation since most of the image judging were incomplete and some inaccurate due to its recentness, however, the website seems promising and might help for future projects.

B. Our dataset

My supervisor provided me with a dataset of images of a flag, with a colourful texture on it, being waved. The images rendered on Blender were frames taken while the flag was affected by wind. Its material varied as well as the fixed points in order to create different folding possibilities. For the first part, there was only one point light source and then we operated on images where the light source could travel inside the front half of the top hemisphere, creating different shading possibilities. The chosen texture was helpful due to its irregularities in colors and the presence of a big black cat in the center which the model could confuse with the background. The dataset was huge however, it contained 27 folders of 2352 frames which adds up to a total of 6.9GB so it surely couldn't fit in the memory. We implemented a directory crawler that would search in the original images directory and find the corresponding albedo and shading in the other directories. The downside being that the training speed got affected.

IV. Model Architecture

We went trough four main models during the semester each improving on the previous one.

A. ResNet-like model

We started with one similar to the first half of the network used in [5]. However we couldn't fit as many residual blocks as them into the GPU memory so we limited it to 5 residual blocks. This architecture is more resistant to the vanishing gradient problem. The model started with a convolution layer, then 5 blocks (containing each a convolution layer, a batch normalization, a ReLu activation, then again an convolution and a batch normalization then the input of the block is added to the current flow and the result gets activated with ReLu) and finally a convolution layer with a linear activation and 3 channels as output. The pixels at the output of the model wasn't limited between 0. and 1. because we wanted to keep the last activation as the linear function so we don't lose the values of the gradient on negative values or very large ones. So we had to clip the output, and then divide the image by the predicted shading, then clip the answer again to get the predicted albedo. The optimizer used is RMSProp and the loss function is the MSE between the predicted shading and the albedo.

Figures 1 and 2 show the results obtained after training the model. The 3 images on top are the respective ground truth image, shading and albedo. Even though it seems like the loss values are fluctuating a lot, their orders of magnitude are around 10^{-3} 10^{-4}. (Plus at that point we were logging the loss results at every step.)

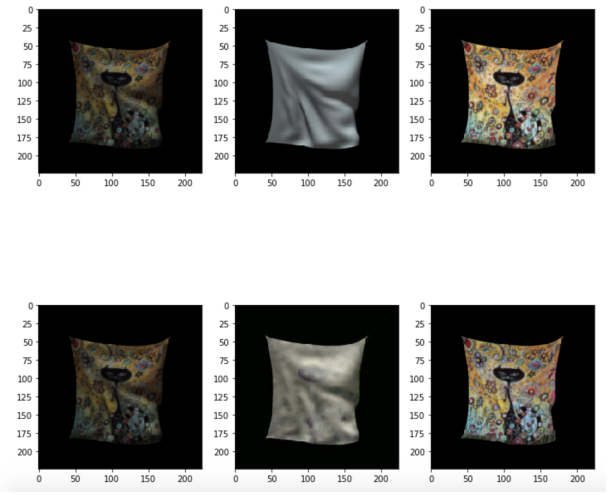

Fig. 1. Results of the first model. Top images are the groundtruths and Bot are predictions. From left to right: Image, Shading, Albedo. Albedo is predicted in a post-processing step by dividing the groundtruth image by the predicted shading. The predicted image is just the multiplication of the predicted shading and albedo.

B. Adding a second output

As we can see in figure 1 in the second image on the bottom row, the predicted shading was giving a "yellowish" feeling, especially after clipping the shading and calculating the albedo, so we wanted to solve this. We realized that adding another loss function, for the albedo prediction would be a good way to limit this. So we had to include the division process into the network. We coded a Divide layer which extends the Keras Merge layer but values were exploding when at least one channel of a pixel in the predicted shading is equal to 0. And that was surely happening often enough because the background is black. So we had to hack around that and add a second output to our model, an "epsilon" layer consisting of a dummy image where all pixel values are 0.001. We pass the predicted shading through a ReLu activation to eliminate negative values and then add it to "epsilon" and divide the image by the sum. The answer is given to a second MSE, this time between it and the ground truth albedo and the two loss functions are averaged. Usually during training we ran the model while only taking into account the first loss function to get the weights stable enough and reduce the amount of zero pixels at the shading

Fig. 2. Plots of the values of the MSE over the time in hours. Top for the validation set and bot for the training set. The values are fluctuating a lot mainly because we let Tensorboard print an event log at every step, but after just half a dozen of minutes the loss dropped to 10^{-3} so these fluctuations are in the same order.

prediction and then compile the model again with 50-50 weights on the loss functions. As seen in figure 4, we now have 2 loss functions, one that calculates the MSE between the predicted shading and the original shading, and one for the albedos.

C. Taking the gradients into consideration

As we can see in figure 3, we got rid of the yellow glow but as before, we still have the outline of the black cat visible in the shading predictions. Hoping to solve this, we got inspired by the loss function used in [5]. They didn't only calculate the MSE for the images, but also the MSE between the gradients of the predictions and the ground truth. This should help us here since the most flagrant mistakes are the edges in the texture. To do this, we passed each output through a linear activation that is not trainable, this way we can duplicate them, and then we use the same ground truth to calculate a "Sobel loss" which consists of creating a (3, 3, 1, 2) a tensor that contains the X and Y gradients of the image. The loss value is then the MSE between the tensor calculated from the prediction, and the one from the ground truth. The plots for the loss values can be seen in figure 6.

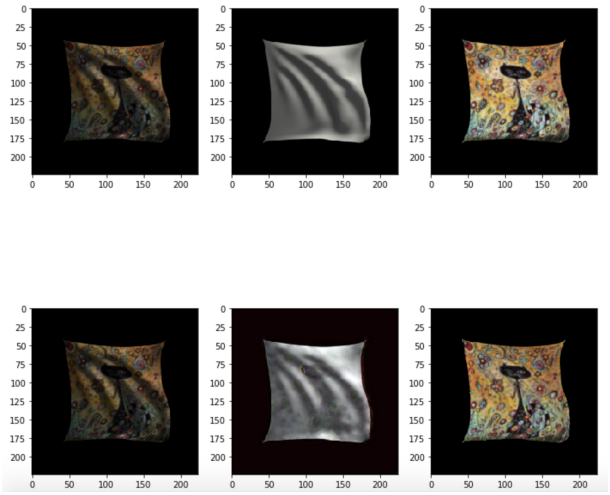

Fig. 3. Results of the second model. Unlike to the case in figure 1, this model has two outputs. Both the shading and albedo are predicted in the model then the predicted image is obtained in a post-processing step by multiplying the two others.

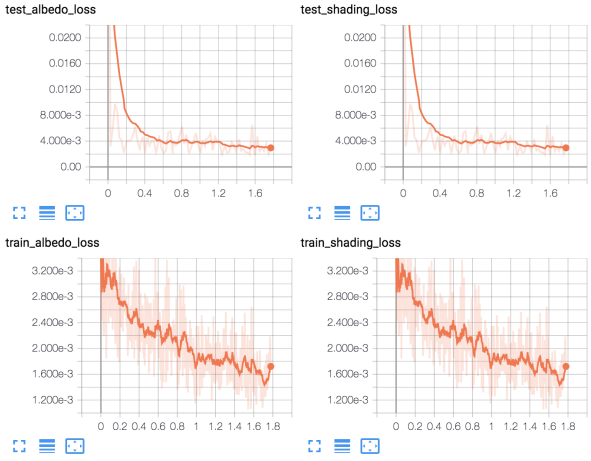

Fig. 4. Loss values obtained during the training of the second model. Again, the losses are plotted over time in hours of training. It might seem that the loss over the albedo MSE is reaching a plateau, but if you look closely, between time 0.8 and the end, both the albedo loss and the shading loss got nearly halved. This could be explained that at first, it takes quite some time to get the albedo predictions right, with more degree of freedom available to the model since colors have a wider range.

In addition to the previous case, this model also has a loss value for each tensor of gradients (albedo and shading for both training and validation) which makes a total of 8 losses. We're only showing 2 here, the average for the training loss and the one for the testing.

D. Giving perceptive field to the neurons

As we can see in figure 5, we mostly got rid of the cat outline. However, all this training and testing was done on images rendered in a scene where only one light is shining from the same position. We weren't sure if the model would understand the same curves in the flag if the light wasn't in the same place. So we tested again on a dataset where the light wasn't fixed and we got the results in figure 7, we got

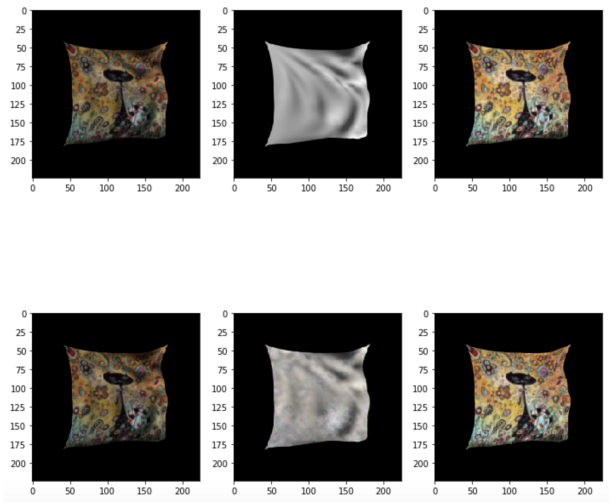

Fig. 5. Results of the third model. Similarly to 3 both the shading and the albedo are predicted in the model.

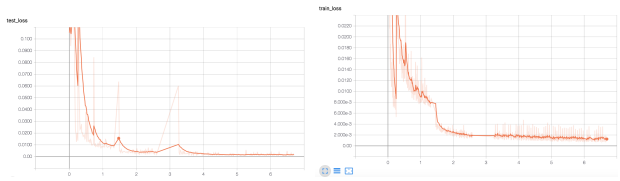

Fig. 6. Loss values obtained during the training of the third model. Some discontinuities are seen but this is because the training is stopped, then recompiled with different loss weights.

the outline of the cat again!

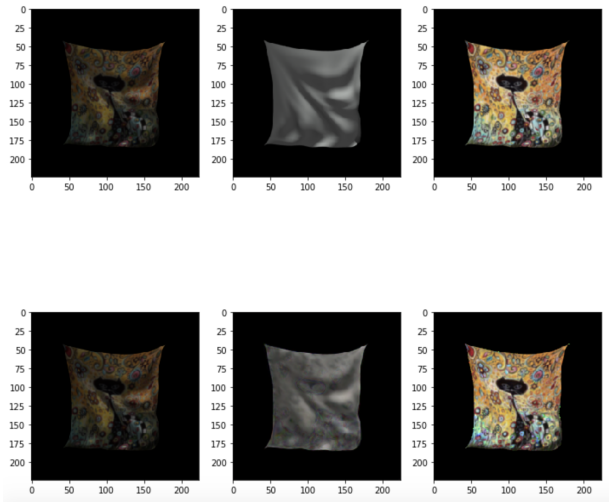

Fig. 7. Results of the third model on a dataset with a moving light. Unlike figure 5, the light in the rendering of the images wasn't always at the same spot.

We knew that in order to solve this, we had to give more perceptive field to the pixels in the feature maps. This is usually done by max pooling or average pooling or convolving with strides. However, this could be complicated with a ResNet like architecture since this would mean the resblocks would have to be customized each to accept feature maps of smaller and smaller width and height. After some research, we saw that SegNet[1] acts similarly to what we're thinking. So what we did is implement a lighter version of their model: 4 encoding blocks where the depth of the feature map doubles every time, and the height and width get halved three times. And then the other way round, 4 decoding blocks to get back the same output as the third model.

The results are far clearer and accurate than the previous model, as shown in figure 8 and the corresponding loss values in figure 9.

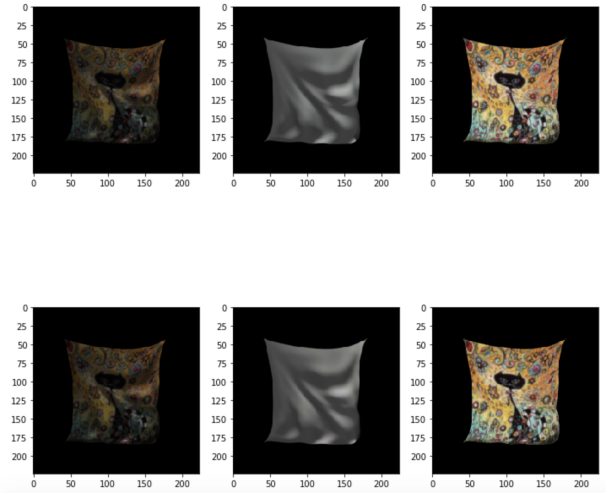

Fig. 8. Results of the fourth model. The difference between the groundtruth, top, and the prediction, bot, can now be hardly seen with the naked eye.

V. Testing on public datasets

With successful results on our dataset, we wanted to see if fine-tuning the model on the few images of the MIT dataset would also be successful.

Figure 10 shows the prediction of the model on a validation image from the MIT dataset.

We fine-tuned the model on the images and got the result in figure 11.

However the MIT are not respecting the $I = A*S$ equation so we also re-synthesized the images by multiplying the shading and albedo components and fine-tuned the model on the new mini dataset and got the result in figure 12.

The results in figures 11 and 12 are clearly not satisfying but if we look at figures 13 and 14 we see that there is a huge improvement compared to just training the model on the miniset.

VI. Conclusions

So in this report, we explained our choice of dataset and introduced the four models we worked with, it is clear that the last one is the most successful one. Even though it wasn't able to learn the MIT dataset very well, it very probably might have over-fitted due to the small size of the dataset, it still showed that our dataset was helpful in giving a first general base knowledge to the model which could be fine-tuned, and probably with more images than

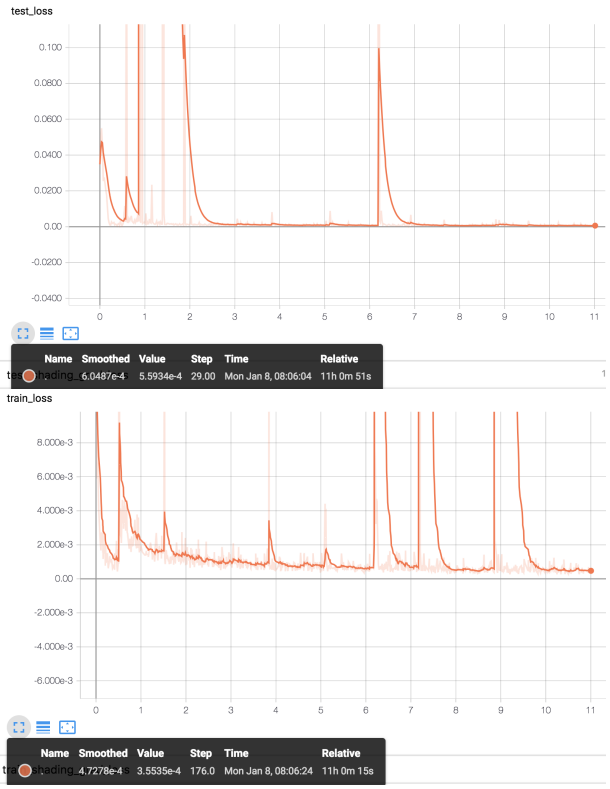

Fig. 9. Loss values obtained during the training of the fourth model. Top is the average of the losses (shading, albedo, gradient of the shading, gradient of the albedo) during testing and bot is during training. After 11 hours of training, the respective losses got to 5.59^{-4} for testing and 3.55^{-4} for training.

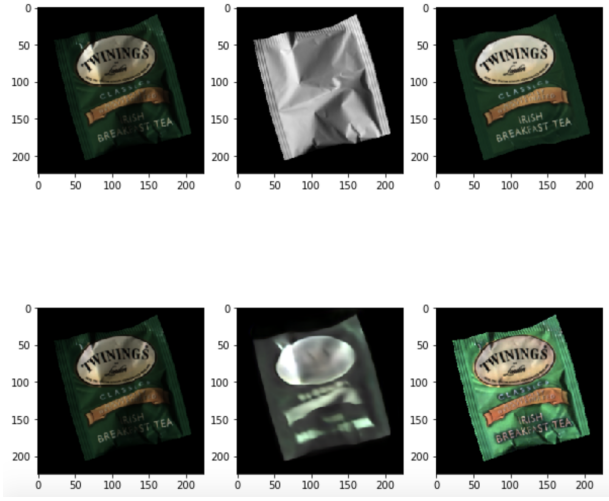

Fig. 10. Results of the fourth model on an MIT image. The model hasn't been trained or fine-tuned on the MIT dataset.

the MIT dataset, could have achieved better results. As seen in figures 11 and 12, the important shadow traces are being detected, so it could just be that the model needs to encounter a few more shapes in order to learn how the shading interacts with them, to this point it has only seen the same flag over and over again and 18 other shapes.
To help that even further, we chose some textures that could

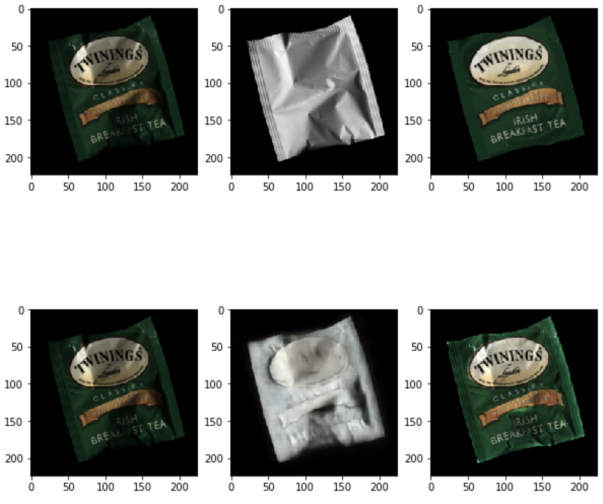

Fig. 11. Results of the fine-tuned model on an MIT image. The prediction seems to have improved considerably compared to figure 10

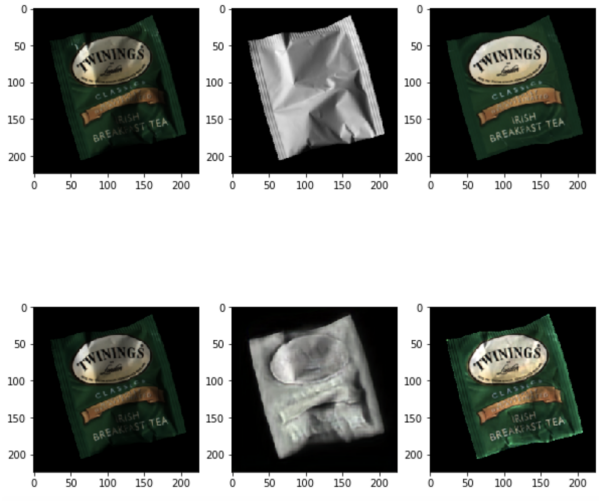

Fig. 12. Results of the fine-tuned model on an MIT synthesized image. The prediction is also done on a re-synthesized image.

be helpful for a future project, or for the continuation of this one. The textures should be applied to the cloth, and they imitate common patterns in either nature or clothes: Two areas we always kept in mind for this project, being able to change the color of cloths in a scene is one of the most likely application, albedo extraction from an image or being able to extract 3D shape of objects mainly with the help of the shading.

VII. ACKNOWLEDGEMENT

The code I left over should be modular enough to allow further improvements, I suggest using the IIW dataset once it is robust enough. If you need any documentation or help in the usage of my scripts feel free to contact me in the next quarters. Thank you CVLAB and especially Jan Bednarik for your guidance.

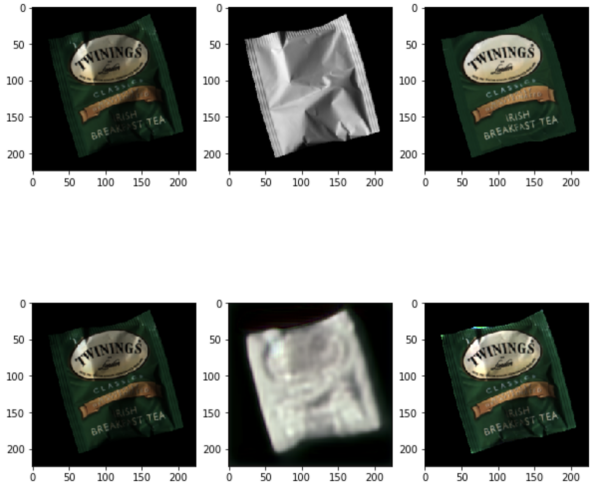

Fig. 13. Results of the model trained only on MIT images.

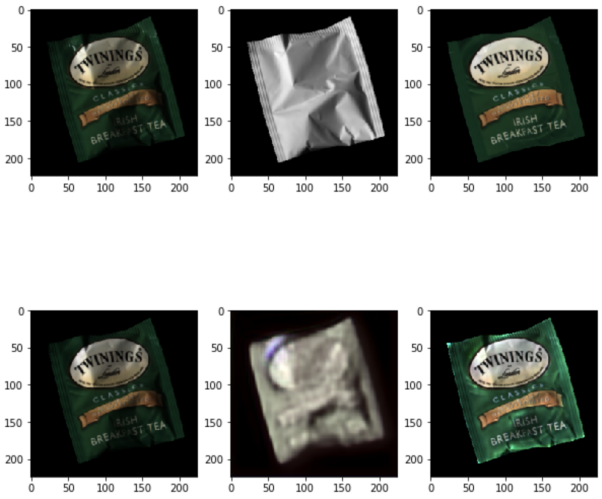

Fig. 14. Results of the model trained only on MIT synthesized images.

REFERENCES

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015.

[2] Sean Bell, Kavita Bala, and Noah Snavely. Intrinsic images in the wild. *ACM Trans. on Graphics (SIGGRAPH)*, 33(4), 2014.

[3] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In A. Fitzgibbon et al. (Eds.), editor, *European Conf. on Computer Vision (ECCV)*, Part IV, LNCS 7577, pages 611–625. Springer-Verlag, October 2012.

[4] Michael Janner, Jiajun Wu, Tejas D. Kulkarni, Ilker Yildirim, and Joshua B. Tenenbaum. Self-supervised intrinsic image decomposition. *CoRR*, abs/1711.03678, 2017.

[5] Louis Lettry, Kenneth Vanhoey, and Luc Van Gool. DARN: a deep adversial residual network for intrinsic image decomposition. *CoRR*, abs/1612.07899, 2016.

[6] Chen Li, Kun Zhou, and Stephen Lin. Intrinsic face image decomposition with human face priors. September 2014.

[7] Takuya Narihira, Michael Maire, and Stella X. Yu. Direct intrinsics: Learning albedo-shading decomposition by convolutional regression. *CoRR*, abs/1512.02311, 2015.