TIAGO KOJI CASTRO SHIBATA
HENRIQUE CASSIANO SOUZA BARROS

ColorMotion: Automatic video colorization

São Paulo
(December 2018)

TIAGO KOJI CASTRO SHIBATA
HENRIQUE CASSIANO SOUZA BARROS

ColorMotion: Automatic video colorization

Monograph of the capstone project of the Computer Engineering bachelor - Escola Politécnica da Universidade de São Paulo

São Paulo
(December 2018)

TIAGO KOJI CASTRO SHIBATA
HENRIQUE CASSIANO SOUZA BARROS

ColorMotion: Automatic video colorization

Monograph of the capstone project of the Computer Engineering bachelor - Escola Politécnica da Universidade de São Paulo

Area of Concentration:
Computer Engineering

Supervisor: BRUNO DE CARVALHO ALBERTINI

São Paulo
(December 2018)

Name: SHIBATA, Tiago Koji Castro; BARROS, Henrique Cassiano Souza
Title: ColorMotion: Automatic video colorization

Monograph of the capstone project of the Computer Engineering bachelor - Escola Politécnica da Universidade de São Paulo

Monograph submitted to the department in:

Authors:

Prof. Dr. Bruno de Carvalho Albertini
(Supervisor)

Tiago Koji Castro Shibata
(Student)

Henrique Cassiano Souza Barros
(Student)

"It is the supreme art of the teacher to awaken joy in creative expression and knowledge"
(Albert Einstein)

Abstract

Given a grayscale video, this monograph proposes a method for colorization using convolutional neural networks in an encoder-decoder architecture. Unlike current image colorization models which, when applied frame by frame, create inconsistencies between frames, this model maintains state in the encoded features and learns to colorize while maintaining the colors of regions that existed on previous frames. Dense optical flow is used to track movement and propagate colors between frames. The method is user-guided in the sense that an user can input pixel colors as additional information, biasing the algorithm at multimodal color choices. Training was done on large scale datasets taken from open source videos and the ImageNet dataset, which was artificially augmented to generate frame sequences with motion.

Keywords: CNN. Colorization. Video. Restoration. Encoder-decoder.

Resumo

Dado um vídeo em escala de cinza, nosso trabalho propõe um método de coloração utilizando redes neurais convolucionais em uma arquitetura encoder-decoder. Diferentemente de modelos atualmente disponíveis para coloração de imagens que, se aplicados frame a frame, criam inconsistências entre frames, nosso modelo mantém estado nas *features* encodadas e aprende a colorir mantendo as cores de regiões que existiam em frames passados. Fluxo ótico denso é usado para acompanhar movimento e propagar cores entre frames. O método é guiado por usuário no sentido de um usuário ser capaz de dar cores de *pixels* como entrada adicional, levando o algoritmo à cor desejada em escolhas de cor multimodais. O treinamento foi feito em bases de dados de larga escala tomada de vídeos de licença aberta e da base de dados *ImageNet*, que foi aumentada artificialmente para gerar sequências de frames com movimento.

Keywords: CNN. Colorization. Video. Restoration. Encoder-decoder.

List of Figures

List of Tables

List of Initials and Abbreviations

API *Application Programming Interface*. 17

APoZ *Average Percentage of Zeros*. 29

CIE *International Commission on Illumination*. 17

CNN *Convolutional Neural Network*. 5, 6, 7, 12, 14, 15, 16

DNN *Deep Neural Network*. 14, 15, 19, 21

GPU *Graphics Processing Unit*. 17, 20, 27

HVS *Human Visual System*. 17

LSTM *Long Short-Term Memory*. 22

MSE *Mean Squared Error*. 18, 20, 21, 24, 28

ReLU *Rectified Linear Unit*. 20

SSIM *Structural Similarity*. 18, 19

Contents

1 Introduction

In this monograph, we propose a method for automatic video colorization, using a machine learning-based approach.

Previous work (ZHANG; ISOLA; EFROS, 2016) has used CNNs trained on large-scale datasets to colorize pictures successfully. However, when applied frame by frame on videos, the predictions have inconsistencies between frames, since no state is maintained between frames and different colors are predicted on the same object after small changes in the input grayscale image.

Since the colorization problem is multimodal, that is, multiple colors are plausible colorizations for a single object, the proposed model can also be user-guided. The algorithm can take input colors for a given object (e.g. the user can choose a t-shirt's color to be red) and will colorize the object and propagate it to following frames.

1.1 Motivation

The problem of colorization in the field of machine learning is one of major interest. It is not a solved problem and poses challenges unlike others, since its solutions are multimodal (the same object has multiple plausible colorizations). Previous research showed that generating believable colorizations is hard, since commonly used objective functions will infer conservatory colors in multimodal regions to minimize the loss, generating images with too low saturation to be convincing (ZHANG; ISOLA; EFROS, 2016).

Our work automates a manual, boring and expensive process: professional picture and video restoration services require expertise and lots of manual inputs from a trained professional. Restoration of videos costs upwards of thousands of dollars per minute: on a quote done with a prominent company in the sector, the restoration of a 9 minute video using their lower quality tier, whose colorization is less convincing, would cost USD 23,000.00.

1.2 Model overview

We take colorization of videos to be an extension of the colorization of images. We propose the use of state-of-the-art machine learning algorithms to colorize videos, prioritizing methods to maintain consistency of colors between frames in a scene whilst correctly colorizing new objects in a frame and detecting frames associated with a new scene. Our base model architecture is heavily based on previous image colorization work, with added state and an intermediate dense optical flow step. The intermediate dense optical flow

step using the Lucas-Kanade method makes the model non-differentiable and, thus, not an end to end trainable CNN; however, we achieve good results training the encoder and decoder separately.

1.3 Evaluation

Results are trained on large scale data using open source videos, personal videos collected by the authors, and ImageNet images augmented to have artificial motion. The results are then tested on people: Volunteers were asked to rate scenes as original or computer colorized, so we can have a metric on how believable the colorization is.

1.4 Justification

As stated in Section 1.1, the traditional process of coloring images involve manual inputs from the user and use of advanced professional editing tools. The same can be said to the colorization of videos, with the added difficulty of maintaining consistency through frames. Some tools can help tracking the motion and colors between frames, but the initial input and challenges involving rapidly changing scenes add to the complexity of the process. These tools often are integrated with other editing software, such as the Mocha plugin (MOCHA, 2018) and the SilhouetteFX Suite (SILHOUTTE, 2018), being utilized as general purpose tools, mainly for color correction.

2 Related work

Most previous works were concerned with the problem of coloring images. One traditional method to resolve this problem was inaugurated by Levin *et al.* (LEVIN; LISCHINSKI; WEISS, 2004), where colorization is achieved through quadratic cost-functions to determinate the color of neighbouring pixels, and the initial colors are given by scribbles input by the user. This user-based approach has received improvements in following works (HUANG et al., 2005) (KUMAR et al., 2012), but the necessity of guidance through user input has always represented a compromise in the process.

One method to simplify the scribble-based approach is the reference-approach, were a reference image is supplied rather than local colors. This method is explored by Kumar *et al.* (KUMAR et al., 2012), where pixels are grouped for both feature extraction and color classification, which in turn is resolved via a randomized decision forest. The reference-approach method is also employed in other problems in the image domain, such as style-transfer (RUDER; DOSOVITSKIY; BROX, 2016).

The first fully-automated technique created was developed by Cheng *et al.* (CHENG; YANG; SHENG, 2015), where a *Deep Neural Network* (DNN) is employed with feature descriptors as the input layer and with UV channels as the output, for each pixel and in the YUV colorspace.

Most state-of-the-art algorithms applied today involve the use of Convolutional Neural Networks (LARSSON; MAIRE; SHAKHNAROVICH, 2016) (IIZUKA; SIMO-SERRA; ISHIKAWA, 2016). The state-of-the-art work in the field applies methodologies such as described in Zhang *et al.* (ZHANG; ISOLA; EFROS, 2016), making use of end-to-end CNNs in an encoder-decoder fashion.

3 Methodology

We adopted five main phases of development: study and choice of existent machine learning algorithms; development of a dataset; specification of the initial neural network architecture; adaptation the architecture to improve model performance and add new features (such as the use of recurrent architectures to maintain state and the use of dense optical flow); and collection of metrics and user feedback based on A/B tests.

3.1 Conceptual aspects

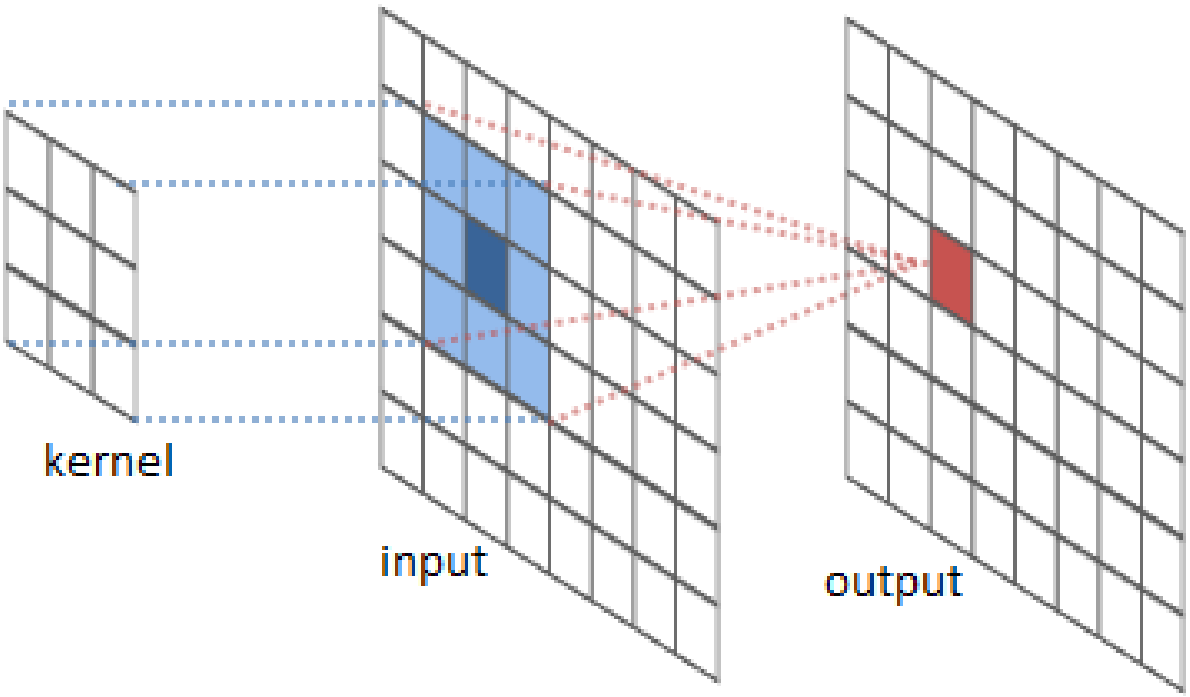
Neural networks for colorization can be divided in two broad categories: those that require inputs from the user, and those that use a automated process. In both cases, the most modern approaches use CNN to leverage different image characteristics, and use those patterns to identify and separate objects in the image and colorize them accordingly. Other approaches are possible, for example through the use of self organizing maps alongside neural networks as shown in Richart *et al.* (MATIAS; JORGE; JAVIER, 2016); but for the purposes of this monograph we will focus in CNN-based algorithms.

A CNN is described by the size and the number of its maps, kernel sizes, striding factors, and the connection table (CIRESAN et al., 2011). Each layer in the network has a number of equally-sized feature maps. To these maps, a kernel is applied, computing the convolution between the filter and the input. This kernel is then applied throughout the map according to the defined striding factor for the available regions of the map. The result of each convolution act as the receptive fields for the neurons in the layer, each having its distinct receptive field.

After the convolutional layer step, it is usual to include a form of subsampling, in order to improve the performance of feature-extraction by reducing the spatial complexity of the input. As higher level features are detected, high resolution details become less and less important, and a more global view becomes more important. Subsampling is an effective form of speeding up the following convolutions by reducing the spacial complexity while also enlarging the effective receptive field of the next layers.

One common method to implement subsampling is through the use of max-pooling layers, where the layer is composed of a small kernel that finds the maximum value in a sub-matrix with the same dimensions as the kernel. This kernel is applied to the input matrix, according to striding (skipping step), similarly to the striding defined in the convolutional layer. Subsampling can also be done on the convolutional layer itself. For instance, Iizuka *et al.* (IIZUKA; SIMO-SERRA; ISHIKAWA, 2016) applies a stride of 2 in order to downsample.

Figure 1 – CNN operation from River Trail documentation.



Source: (LABS, 2018)

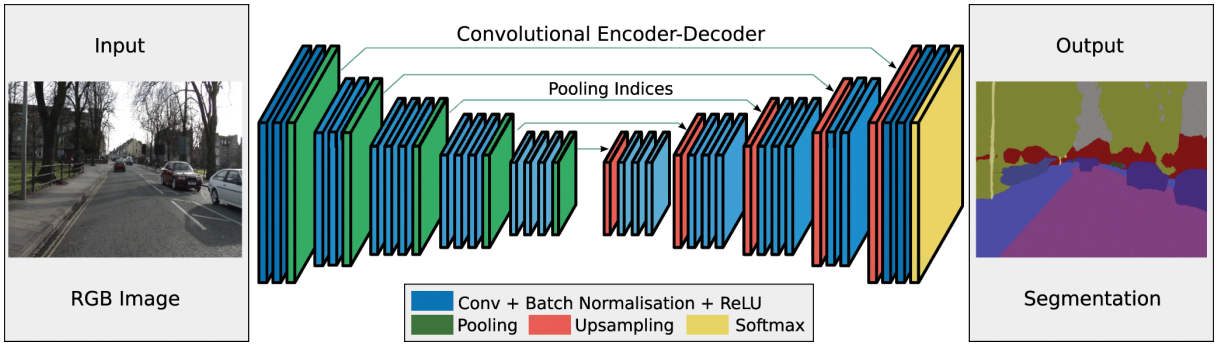
Fully convolutional DNNs were used throughout this work to take advantage of its space invariance and local correlation between pixels in an image.

Encoder-decoder architectures are heavily used. The architecture is often used in images when the task is related to image translation (such as transforming an image into a semantic segmentation, colorization and styling).

The encoder is made of a series of convolutional layers with subsampling. The first layers learn to detect low-level features of the image (such as contours and colors), while deeper layers learn higher and higher level features, as is usual in CNNs. Thus, deeper layers have lower spacial resolution but are wider (higher number of filters).

The decoder has transposed convolutions to merge the information from multiple filters and re-scale the image. Skip connections are also used to pass lower level details to the decoder layers.

Figure 2 – Encoder-decoder architecture.



Source: (PANBOONYUEN, 2018)

Another important aspect in image processing regards the choice of color space, as each one carries advantages and disadvantages. It is possible to organize existing color spaces in three categories (TKALCIC; TASIC, 2003):

- HVS based: these types of spaces, such as the RGB and HSV color spaces, leverage characteristics of the human biology and color processing mechanisms in order to create their color spaces. They try to emulate the spectrum visible to the human eye, and each specific color space might treat other aspects of color (e.g. brightness, saturation) as secondary. These color spaces, although intuitive to work with, have inherent problems regarding device dependency, discontinuities in the color space and discrepancies between computed lighting and perceived lighting.

- Application Specific: certain color spaces, such as the CMY(K) and YUV color spaces, were created to attend determined demands in applications. The YUV color space, for instance, was created to ensure that color televisions could be implemented while keeping broadcasts compliant with older, luminescence-based models. These color spaces can present interesting properties relevant to image processing. For instance, Cheng *et al.* (CHENG; YANG; SHENG, 2015) used the YUV color space as it minimizes the correlation between each component of the color space.

- CIE based: these color spaces, including CIE XYZ and CIE L*a*b*, are defined by the *International Commission on Illumination* (CIE), and are focused on being device independent while keeping a strong correlation with the human perception. These characteristics make it a good system for applications that work with color without or with little input from the user.

3.1.1 Tools and technologies

The project is written in Python, using the Keras framework with the Tensorflow backend. In addition, the OpenCV and Numpy libraries are also used. Lastly, we used CUDA

and Docker as supporting technologies. A small summary of technical tools and their usage is:

Table 1 – Technical tools.

Tool	Usage	Website
Python	Programming language.	https://www.python.org/
CUDA	API for GPU-based network training.	https://developer.nvidia.com/cuda-zone
Keras	High-level API for neural network development. Requires a back-end, such as TensorFlow.	https://keras.io/
OpenCV	Image processing package, used for dataset processing and for its implementation of optical flow.	https://opencv.org/
Numpy	Numeric processing package, used as a prerequisite for Keras and OpenCV, manipulation of OpenCV matrices and network weights.	http://www.numpy.org/
Docker	Container software used for portability and easy deployment.	https://www.docker.com/

Source: authors.

3.2 Dataset

3.2.1 Colorspace

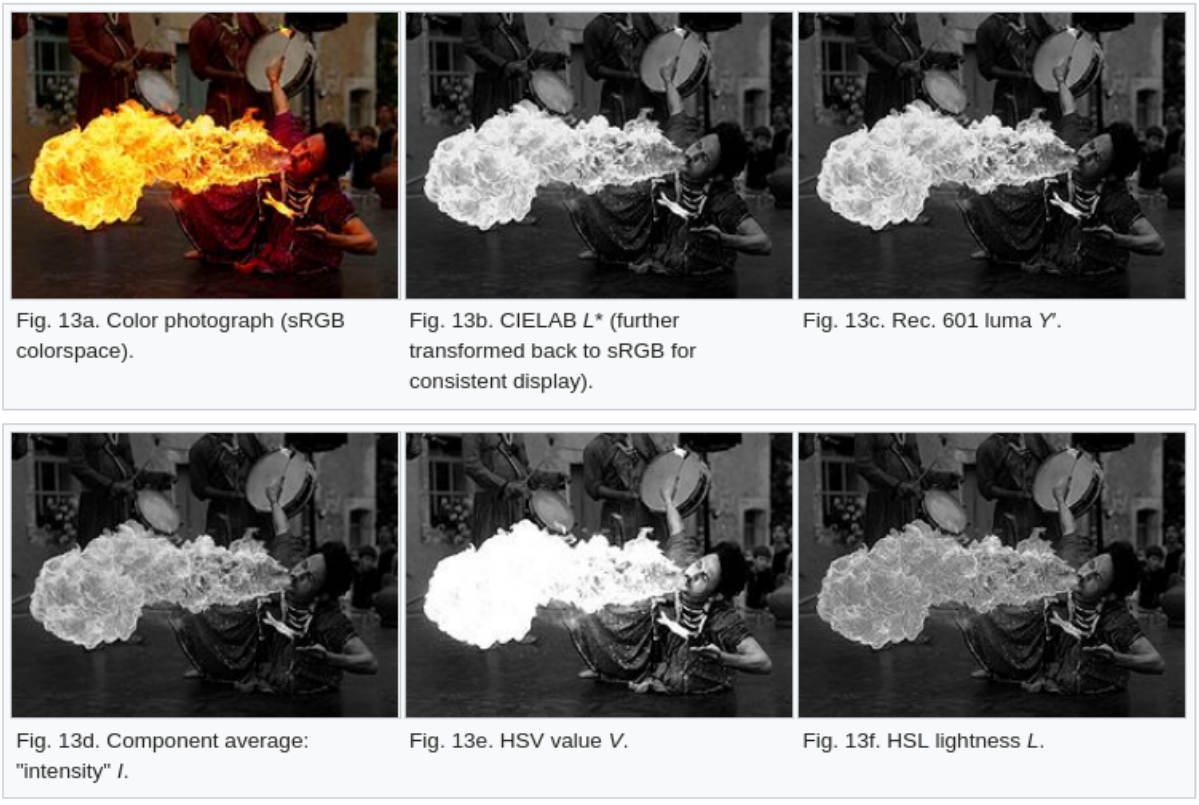
As stated on Colorful (ZHANG; ISOLA; EFROS, 2016) and seen on related research, the L*a*b colorspace is most suitable for the colorization task. Its L* channel is the achromatic lightness quantity, which depends solely on the perceptually achromatic luminance. Thus, the perceptual lightness of a colorized image to a human observer appears most similar in lightness to a grayscale image when using the L*a*b* colorspace and maintaining the L* value. We use all network input images as a single channel L* image, and all color outputs are given in the a*b* colorspace.

3.2.2 Processing of scenes from movies

We wanted a dataset made of open source content, with permissive licenses. After some research, we decided to use the open source movies Tears of Steel (HUBERT, 2018) and Valkaama (BAUMANN, 2018) as a starting point.

To split the movies into independent scenes, two strategies were employed. We created a program to compute the *Structural Similarity* (SSIM) index between each pair of frames and, whenever the SSIM was high, we assumed a new scene was starting. The SSIM is

Figure 3 – Comparison between different color spaces.



Source: (BELGIUM, 2018)

similar to the simpler, less robust *Mean Squared Error* (MSE) metric, except it is more resistant to noise and compression artifacts.

We also ran a histogram analysis on the pixel values of each frame and discarded frames without enough variability. This strategy removed introduction frames, black frames from fading effects, final credits and other undesired frames whose content had very low variability.

We implemented a program to parse the dataset and run both analysis in OpenCV. A few scenes containing too fast movement yielded a high SSIM and were mistakenly cut into multiple scenes. They had to be manually inspected and fixed. After manual inspection, the dataset had 142753 frames in 836 distinct scenes. This small dataset was used as a starting point.

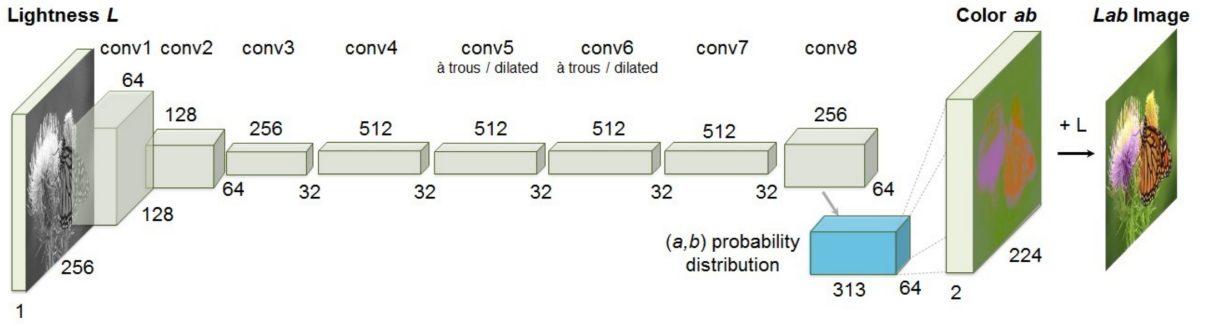
The dataset structure was exported to files describing ranges of frames corresponding to each scene. This metadata was written in JSON format to make re-creation of the dataset easier in other machines. It's avaiable on (SHIBATA; BARROS; TANAKA, 2018a).

3.3 Baseline model

Our first, baseline model was created using a convolutional DNN. We borrowed the same architecture used in Colorful (ZHANG; ISOLA; EFROS, 2016), except we reimplemented it in Keras (the original implementation is in Caffe). Our main goal with this model was to gain familiarity with the tooling we would use in the rest of the work.

The architecture implements an encoder-decoder architecture with skip connections.

Figure 4 – Colorful architecture.



Source: (ZHANG; ISOLA; EFROS, 2016)

Each convolutional layer is a block of 2 or 3 repeated convolutional layers with ReLU activation, followed by a batch normalization layer. No pooling layers are involved and changes in resolution are achieved through spatial downsampling or upsampling between convolutional blocks through strided convolutions and strided transposed convolutions.

To enlarge the receptive field of convolutions with encoded features and empower the network to detect larger features, encoded features go through dilated convolutions.

3.3.1 Objective function

The objective function used in the baseline model is the MSE since it is easy to use. It is given by $MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$, where Y_i and $\hat{Y}_i$ represent the ground truth data and the network output. Related work (ZHANG; ISOLA; EFROS, 2016) shows that MSE is not a good objective function to create convincing colorizations, since it gives very conservative predictions in multimodal objects, creating undersaturated images. Other methods would be more suitable, such as regressing a color probability distribution function similarly to Colorful, but are harder to implement and train.

3.3.2 Environment

We prepared an environment for training in the cloud. GPUs available in servers are more powerful than the ones we own and have more RAM to cope with large models. The chosen cloud provider was Paperspace (PAPERSPACE, 2018) due to its lower cost when compared to other providers.

A Docker container was created so we could prepare the environment and easily port it to other machines. The environment can be found at (SHIBATA; BARROS; TANAKA, 2018b), and contains NVIDIA CUDA libraries, Keras, TensorFlow, Python, and other project dependencies. The container is run with nvidia-docker (ABECASSIS, 2018), a project that allows containers to access NVIDIA GPUs so that CUDA applications can be containerized.

3.3.3 Implementation and training

We reimplemented the model in Keras successfully and ran it locally and remotely using the Docker container. We used Keras' utilities to show a summary of the model and plot its blocks and layer count for each block. The model graph can be seen in Figure 15. Layers that were unavailable in Keras (e.g. Caffe's Scale layer) were replaced with *Lambda* layers, which we implemented ourselves.

The model worked and we could start training it. As it is known in DNN training, starting from scratch with random weights is not recommended, because training will take a very long time to converge. Instead, a model should be initialized with trained weights (even if trained on a different domain, since low level image features are shared between most domains and can be transferred). We let it run with untrained weights just to ensure our training environment was working properly, and got an initial MSE loss oscillating around 8000. After a few epochs with the loss oscillating, we stopped the process and decided to go find trained weights for the network.

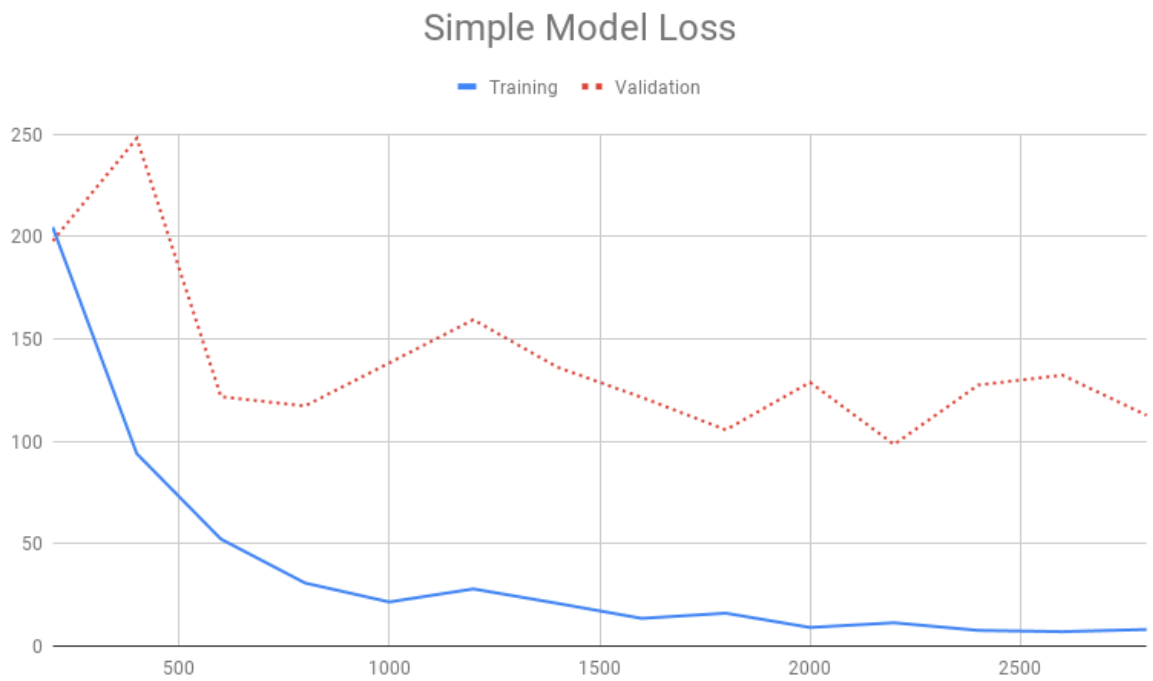
The original weights were in Caffe's binary protobuf format. We could create a program to read the protobuf and export the weights, but instead we chose to use a ready-made converter. The MMdnn (YAO, 2018) project (Model Management for DNN) is a suite of tools that includes a converter between popular formats. Unfortunately, we couldn't use the Caffe to Keras converter directly, since it crashed whenever it found a layer that didn't have a direct Keras replacement (such as the Scale layer). Instead, we used MMdnn to convert the Caffe weights to a Numpy matrix serialized to its binary format. Since we already use OpenCV and Keras (which make heavy use of Numpy), importing the Numpy weights was easier than dealing with the original protobuf weights.

After initial training, this dataset proved too small for the task in hand. In our methodology, we employed a network with a high number of parameters, which is very prone to overfitting. The network did overfit a lot in our initial tests, as can be seen by the loss MSE of 4.21 and validation MSE of 98.43 after 5600 steps (Figure 5).

3.4 Improved models

In subsequent models, we employed the strategy of doing small changes, retraining based on previously trained weights, and keeping improvements. The model was kept large and

Figure 5 – Simple model loss.



Source: authors.

we didn't focus on removing unused weights until later.

3.4.1 Keeping state

We focused efforts on stateful models to maintain consistency between frames. We chose to implement a simple stateful strategy, by feeding the network with information from its execution in the previous frame. Our model can compute its persistent state using *only the previous frame*, not requiring a lengthy window of past frames. Complex stateful architectures and gates, such as LSTM networks (GREFF et al., 2017), were dismissed for being hard and slow to train. Furthermore, we wanted our model to be fast and simple during runtime, so that it could be applied in real time. Other research in the area (CHEN et al., 2017) shows that short-term information propagated frame-to-frame can generate long-term consistency in a convincing matter.

Our architectural choices imply that our model has some limitations. It is incapable of "remembering" the color of objects, should they temporarily leave the screen and then return.

3.4.2 Dataset

A lot of work was done in building a higher quality dataset. We changed the batch generators to generate batches of pairs of neighbouring frames, so that they could be

fed to the stateful model training routines. We implemented custom batch generators that would load a pair of frames from the disk and apply random augmentations to each pair before returning them to the training routine. Data augmentation during training reduced overfitting and improved the loss marginally.

The following operations were randomly applied during augmentation:

Table 2 – Augmentations applied to pairs of frames.

Transformation	Meaning	Range
theta	Random image rotation	[-15°, +15°]
shear	Shear angle	[-20°, +20°]
tx	Translation in X	[-12, +12]
ty	Translation in Y	[-12, +12]
zx	Zoom in X	[0.8, 1.1]
zy	Zoom in Y	[0.8, 1.1]

Source: authors.

The dataset was enlarged with videos from the free video repository Pixabay, under the Creative Commons licence, and from videos in the author's personal archives. The enlarged dataset has 591780 frames in 1629 distinct scenes.

Lastly, when training our last model, we chose to use ImageNet (DENG et al., 2009) data in training. Part of the ImageNet dataset was downloaded from the URLs provided in the ImageNet website. The dataset has a variety of pictures belonging to one of thousands of "synonym sets". We wrote scripts to download the images, ignoring any invalid server responses (many images were removed since they were posted in the dataset), check if the images are valid, and save them to disk. 1071981 valid images were downloaded. We artificially generated flow information, simulating the movement between consecutive frames of a video.

To generate artificial motion data, we took images from the ImageNet dataset and applied minimal distortions to it so that it would artificially have flow when compared to the original image. We reused our data augmentation code, except with much less aggressive changes, since changes between a pair of frames is usually small.

The following operations were applied to artificially generate what could be the image's previous frame, were it part of a video stream:

Table 3 – Augmentations used to generate artificial flow in ImageNet images.

Transformation	Meaning	Range
theta	Random image rotation	[-5°, +5°]
shear	Shear angle	[-10°, +10°]
tx	Translation in X	[-4, +4]
ty	Translation in Y	[-4, +4]
zx	Zoom in X	[0.9, 1]
zy	Zoom in Y	[0.9, 1]

Source: authors.

3.4.3 User guided colorization

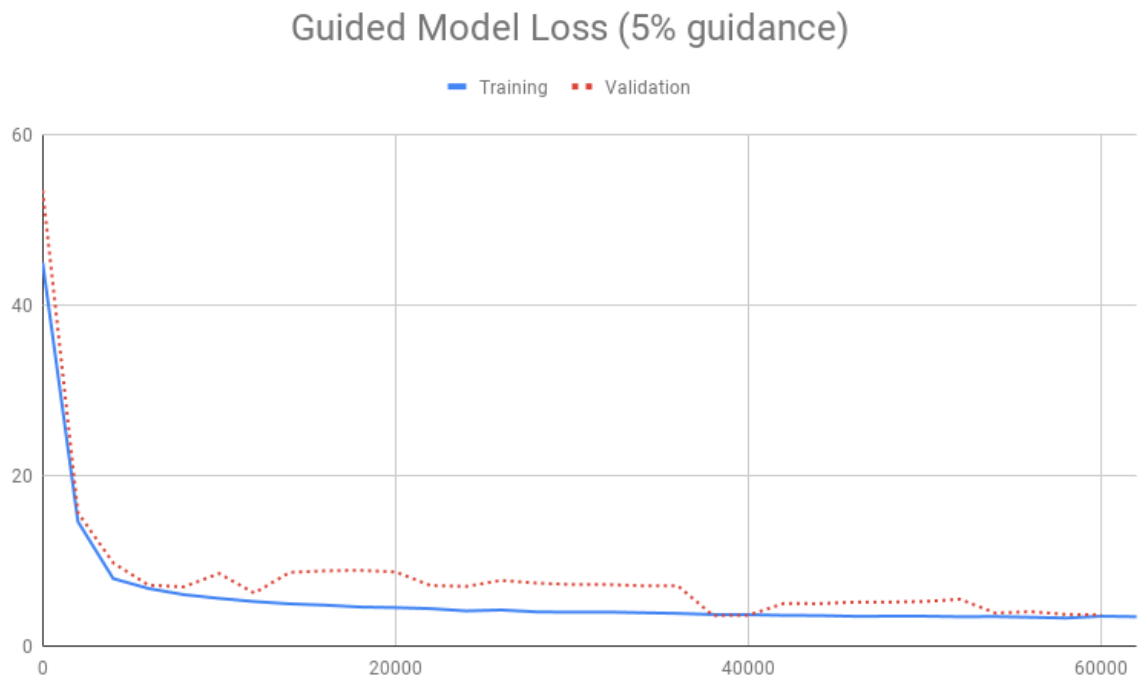
We want our algorithm to support user guided inputs to make it more powerful at multi-modal color choices. The user should be able to select a color for a specific pixel, and the algorithm should use the color as a local hint, intelligently propagating it to neighbouring pixels.

Our user guided model has 3 additional input channels, containing the chosen values for a* and b*, and a binary mask representing whether to use the values (1) or ignore them (0). This setup is based on (ZHANG et al., 2017).

During training, a random mask is generated with a few pixels of the ground truth data, so that the algorithm can learn to use this information for colorization. In our first test, we gave the network 5% of the ground truth values during training. The binary mask was generated randomly with each element having a 5% probability of being 1, and non-zero elements had their a*b* values copied to the network guidance matrices.

The network was able to learn how to copy the given a*b* data (Figure 6). The final MSE loss was 3.9. We realized 5% is too large of a value (equivalent to providing the color of a pixel at about every 4.5x4.5 pixels in the image) and used much smaller guidance in following trainings.

Figure 6 – User guided model loss (5% guidance).

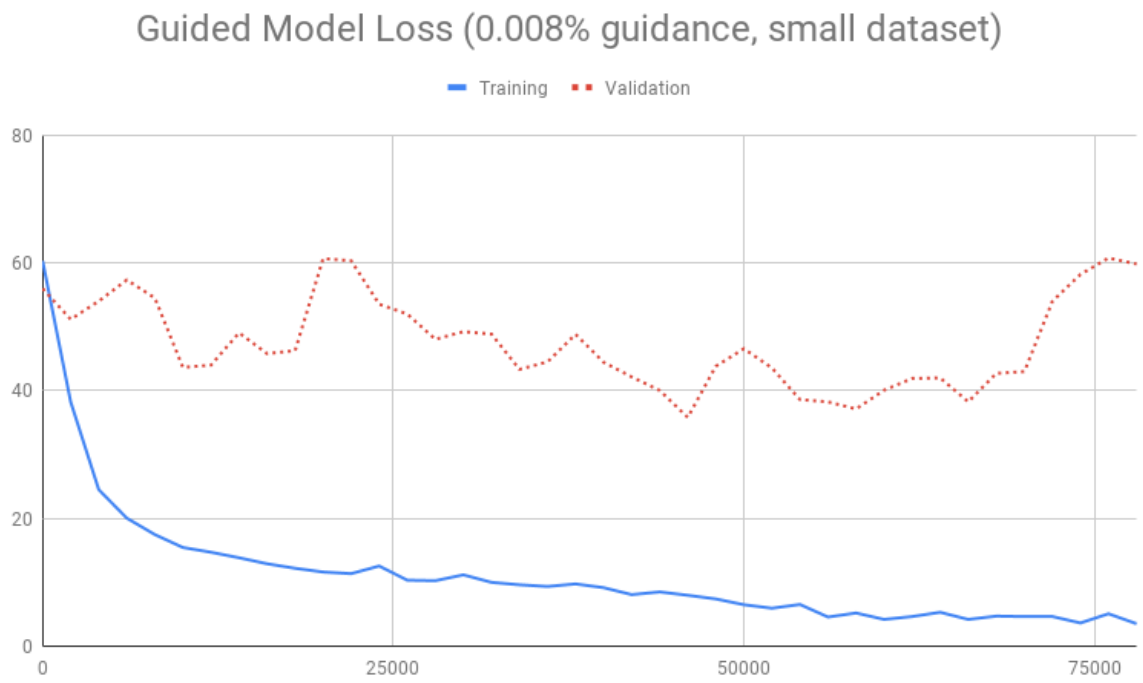


Source: authors.

3.4.4 User guided colorization with less guidance

We trained the network again, but using 0.008% guidance. In a 256x256 image, it is equivalent to giving the network around 5.2 local hints per image. At this point our dataset was still small with 142753 frames from the movies Tears of Steel and Valkaama. It was prone to overfitting didn't have a lot of variation between the training and validation sets. The lowest MSE validation loss achieved was 38.7 7.

Figure 7 – User guided model loss (0.008% guidance).



Source: authors.

3.4.5 User guided colorization with improved dataset

We started working on improving the dataset. Here, we added the Pixabay videos and most ImageNet images before retraining the user guided model.

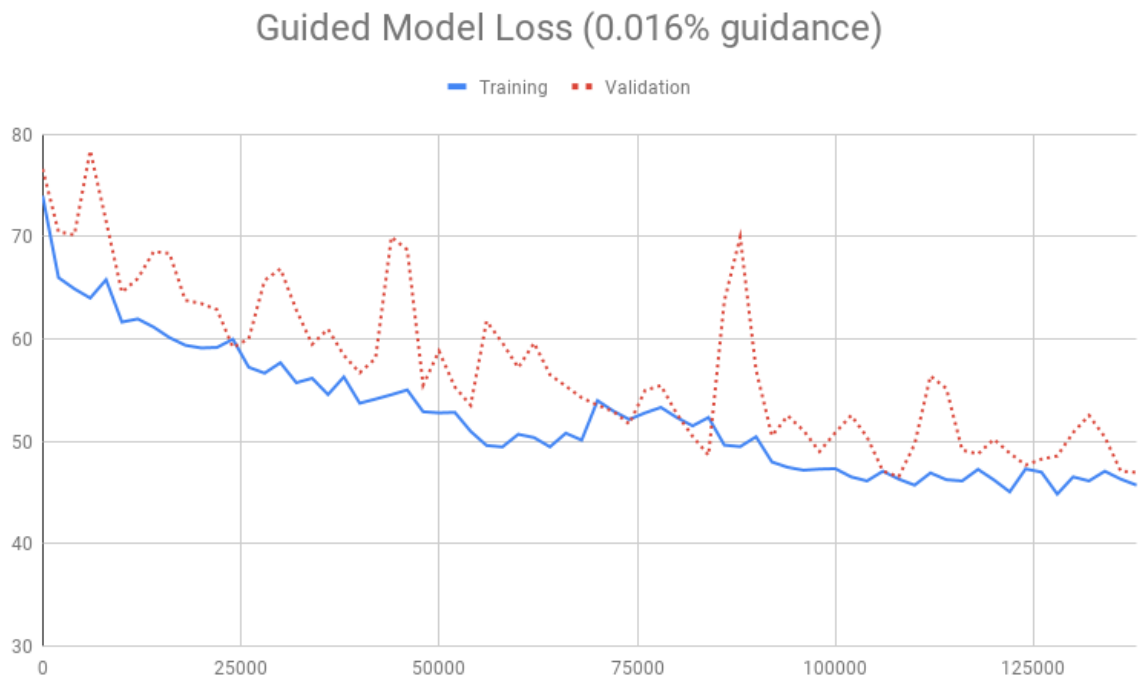
The ImageNet images created a much harder dataset, with a lot of variation, hence the loss is now higher. The final validation loss was at 46.57055 8.

3.4.6 Stateful model

Implementing a stateful model, capable of maintaining consistency between video frames, was our main goal. As mentioned in 3.4.1, we chose a fast, simple stateful architecture.

Our architecture is based on (CHEN et al., 2017), which uses a similar architecture to achieve coherent style transfer (avoiding flickering between frames when transfering style to a video). In this paper, the authors propose the the eencoded features to be

Figure 8 – User guided model loss, with larger dataset.



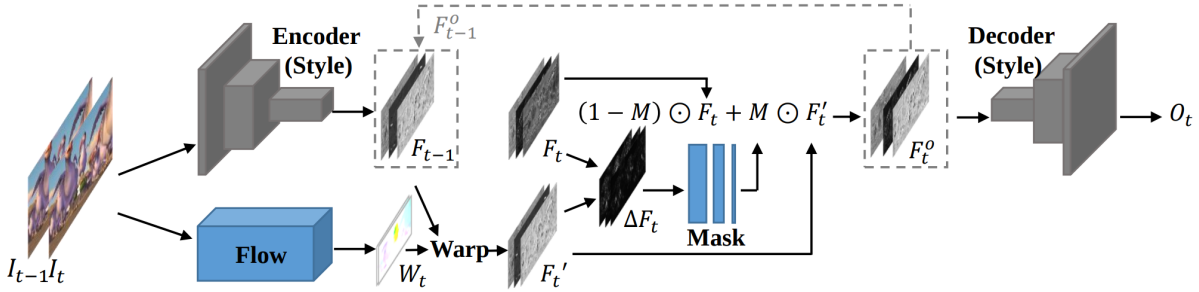
Source: authors.

stateful. Dense optical flow is used to track movement in the scene. For each new frame received, flow is computed using a flow regression neural network. The previous encoded features (F_{t-1}) are warped to generate (F'_t). F_t, the current encoded features, is generated by feeding forward the current frame. Knowing that consecutive frames with an easy mapping between them (without occlusions or motion discontinuities) generate well-behaved flow and can be easily warped to form the next frame, the authors compare F'_t to F_t. If they are similar, the warped features from the last frame (F'_t) can be reused; if they are different, there is likely occlusion, appearance of new objects or motion discontinuities in the new frame, and new colors are generated based on the new frame's encoded features (F_t).

The comparison between F'_t and F_t is done by subtracting them from one another, then feeding the difference into a small network, used to regress a mask in [0, 1] range. Values close to 1 favor the use of F'_t, while values close to 0 favor F_t. F'_t and F_t are linearly interpolated using the mask values.

We implemented a similar architecture in Keras. For simplicity, we use a classical dense optical flow algorithm instead of a flow regression neural network. We chose the Lucas-Kanade differential method, which calculates the optical flow for each pixel by solving optical flow equations for its neighborhood. The authors of the original architecture did not specify their mask regression network structure; we used two convolutional layers with kernel size 3x3 and 16/24 filters, followed by a 1x1 convolution with a single filter, which

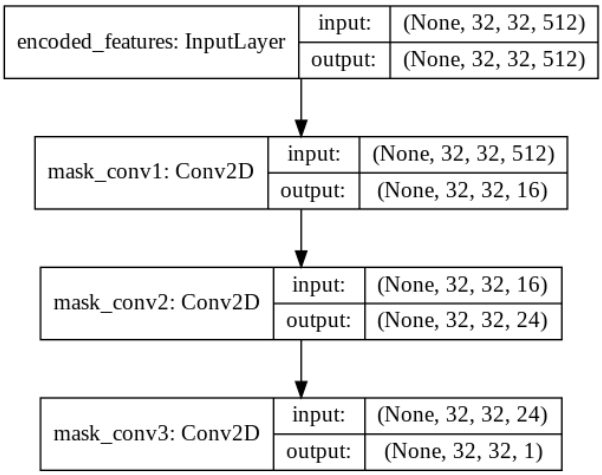
Figure 9 – Stateful architecture overview.



Source: (CHEN et al., 2017)

worked well for us (Figure 10).

Figure 10 – Mask regression network.



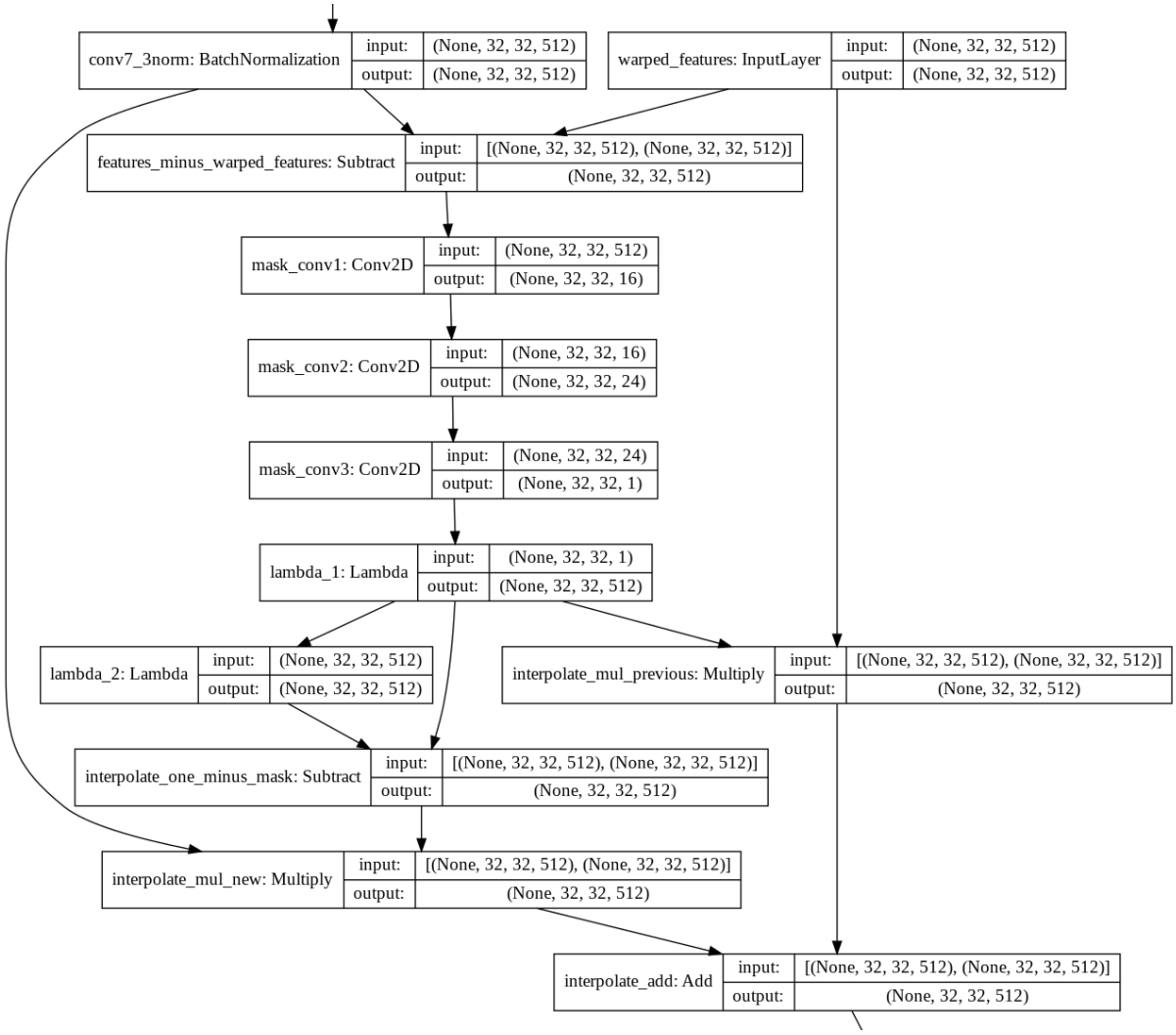
Source: authors.

The stateful part of the model can be seen in Figure 11. The *lambda_1* and *lambda_2* layers were implemented by the authors. *lambda_1* duplicates its single-channel mask input to 512 channels, to make it compatible with the encoded filters depth. *lambda_2* creates a tensor of 1s with the same dimensions as its input.

Training was made much harder due to the optical flow step in the encoded features and the need to specify the decoder inputs after running our optical flow and mask procedure. We had to split the encoder and decoder to evaluate them separately. Splitting and loading two models at once in Keras is not trivial, since the library maintains global state about the currently loaded model, and requires direct backend access. Furthermode we have used the TensorFlow backend, which by default allocates as much GPU memory as possible and uses its own allocator during execution instead of gracefully requesting memory as needed.

We tried to run both models in the same process but hit many problems with Keras, which was not designed with this used case in mind. In the end we executed the encoder and decoder in separate processes using the Python *multiprocessing* library to serialize

Figure 11 – Stateful part of the model.

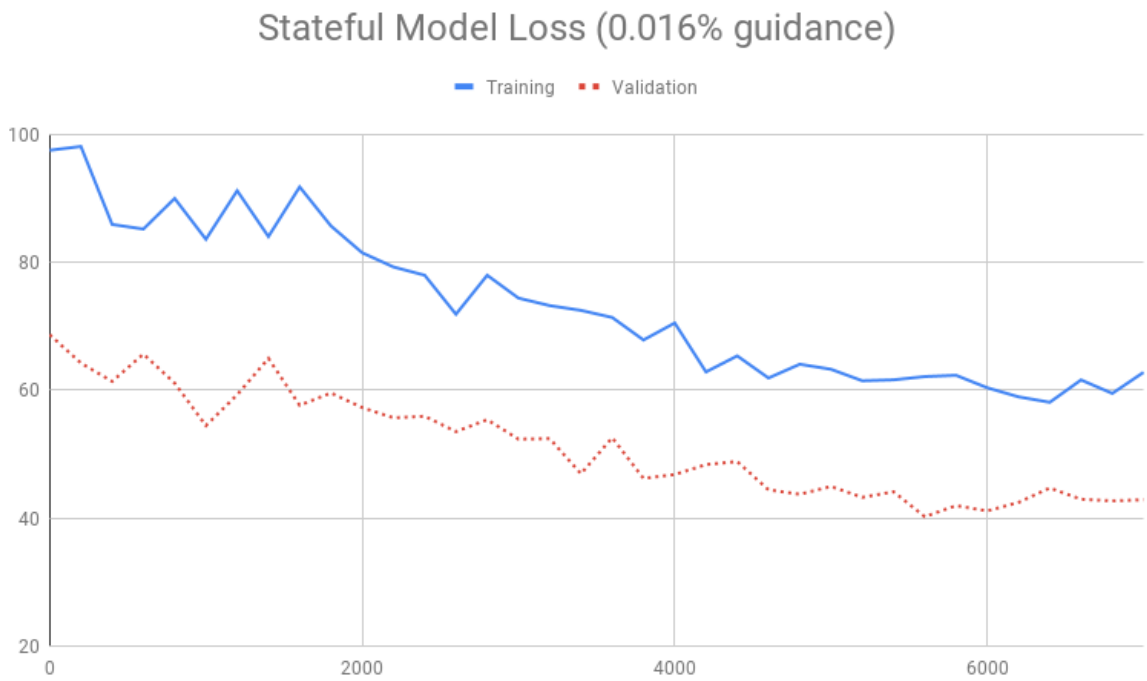


Source: authors.

data for inter-process communication. On each process we manually created a TensorFlow session with *ConfigProto.gpu_options.per_process_gpu_memory_fraction* set to a bit less than half the available GPU memory and set Keras to use that session. All the communication between processes was done by storing data into Numpy multidimensional arrays and sending them with *multiprocessing*.

At this point, we also implemented the generation of artificial flow from static ImageNet images described in 3.4.2. After training, a validation loss of 41.1 was achieved (Figure 12). We had the training loss higher that the validation loss throughout training. We attribute it to the use of augmentation and artificial flow in the ImageNet dataset: we believe our artificial transformations were more significant than the average frame-to-frame changes seen on most videos. Since the validation dataset was only populated with real videos, the training dataset with images with artificial flow became harder to predict.

Figure 12 – Stateful model loss.



Source: authors.

3.4.7 Optimization

After these initial results, we achieved a validation MSE loss of 38.43.

Our initial training strategy was to start with a deep, large model and use model pruning techniques to remove layers that don't affect the result and compress the model while achieving similar performance. To this end, we applied the *Average Percentage of Zeros* (APoZ) method. The method feeds the network with validation data and analyses the output of each layer. After analysing their output with many inputs, filters with overall very low activations are considered unimportant and discarded. The method is important in reducing model size; removing a filter generates a waterfall effect in the rest of the model, since the filter's weights and all weights that would connect to that filter will be discarded.

The keras-surgeon (KERAS, 2018) open-source package was used for pruning weights. It eases the removal of filters and all of its dependencies. Before weight pruning, the model had 34,051,138 trainable parameters, and 390MB when serialized. After pruning, the final network possess 28,879,168 treinable parameters, occupying 111MB of space.

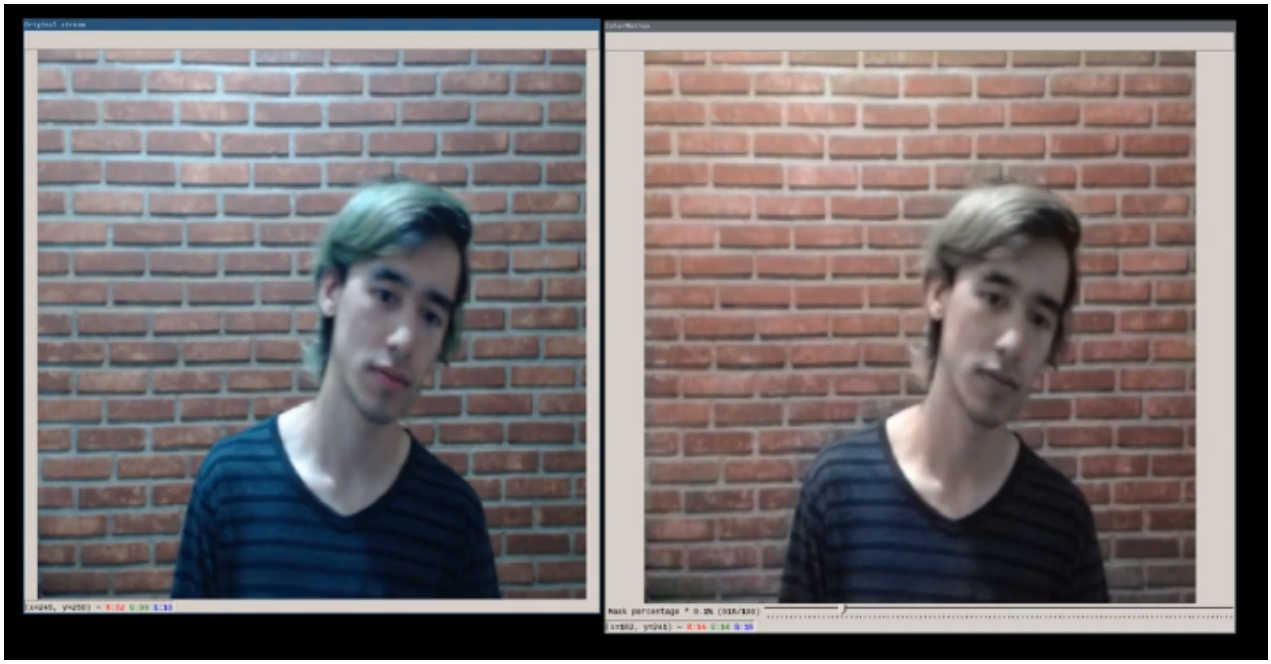
3.5 Performance

Using a GTX1080 consumer level card, streaming the frames one by one (without batching) leads to inference times of 32ms in the unoptimized, large, deep model. Batching

multiple frames yields performance of up to 28ms/frame. These models are applicable in real-time.

A real-time demo was also created, with a GUI program capturing a video stream from a webcam, converting it to L*a*b* colorspace and running the L* channel through the network, with configurable amounts of guidance. The colorized result and the original webcam stream are shown side by side (figure 13).

Figure 13 – Real-time demo using a webcam *(left: original stream, right: colorized stream.)*



Source: authors.

4 Results

4.1 User tests

In order to validate the results obtained by the network, we ran ten seconds videos previously unbeknownst to the network by it. These videos are scaled to 256X256 resolution. 0.008% of the original a*b* values were used as guidance (an average of 5 pixels per frame). The results were then shown to volunteers, who randomly received a series of original or artificially colorized videos. For each video, they were asked to determine if it was original or colorized by a computer. The results of these tests can be shown in 4. Videos were watched 165 times in total.

Table 4 – Results obtained after the AB tests.

	Chosen		
Real	**No**	**Yes**	
No	38.6	11.4	50
Yes	8.6	41.4	50
	47.1	52.9	

Source: authors.

We observed that in the videos colorized by the network, $11.4/50 = 22.8\%$ of volunteers believed that the colors shown in the clip were real. In another test, the volunteers were shown both versions side by side and were asked to identify the clip with the real colors. 23.5% of the time, volunteers where fooled and chose the computer colorized version. Examples of test frames are shown in figure 14, ordered by the percentage of users fooled.

Figure 14 – Collage with percentage of users fooled.

Source: authors.

5 Discussion of achieved results

One of the main results observed in figure 14 was the efficiency of the network in reproducing external environments, specially landscapes, wildlife and vegetation, and conversely difficulty in coloring internal environments, human crafted objects and some specific, less common objects. This can be attributed to the multimodal nature of human crafted objects and environments, which lead to conservative predictions by the network with a low a*b* value. The ImageNet-based dataset might also be skewed, having few internal environments with multiple objects in sight. Better data collection and another objective function could help mitigate these issues.

Improvements would also be likely if we used a neural network for dense optical flow estimation instead of the Lucas-Kanade method. An intermediate neural network for dense optical flow could be trained end-to-end with the rest of the network and possibly achieve better global results. Currently, there are a few options for fast optical flow estimation in deep learning, and our network would likely remain usable in real time. Some examples of these options were observed in Ruder *et al.*(RUDER; DOSOVITSKIY; BROX, 2016), such as Revaud *et al.* (REVAUD et al., 2015) and Weinzaepfel *et al.* (WEINZAEPFEL et al., 2013). This approach is also applied in (CHEN et al., 2017).

5.1 Comparison to competing methods

When compared to image colorization models applied frame by frame, our model has less flickering and gives overall better results. It shows a MSE much smaller than competing models. Individual frame results look less realistic (since the model was optimized to minimize MSE and at times gives conservative, low saturation results), but the overall result in videos is good and can often fool users.

6 Conclusions

In this monograph, we propose an architecture for color prediction in videos, focusing on the maintenance of inter-frame consistency. To this end, we propose a model that maintains the encoded features of the previous frame, alongside the use of dense optical flow. We also propose a color-guided mode, as a solution the multimodal nature of this problem.

Our model can be used in real-time streams and has good performance on videos, showing low flickering artifacts. User A/B tests proved its efficiency in some scenarios. The use of a MSE objective function gives lower quality single frame colorization, since the conservative prediction has low saturation and doesn't look vivid, but generated very low inter-frame artifacts.

The generated dataset and its metadata is open-source and can be used in future research, including future works proposed in this monograph, such as better optical flow implementations.

Bibliography

ABECASSIS, F. *Repository for the nvidia-docker*. 2018. Available from Internet: ⟨https://github.com/NVIDIA/nvidia-docker⟩.

BAUMANN, T. *Valkaama: collaborative Open Source Movie*. 2018. Available from Internet: ⟨http://www.valkaama.com/⟩.

BELGIUM, C. *Jaipur Maharaja Brass Band*. 2018. Available from Internet: ⟨https://en.wikipedia.org/wiki/HSL_and_HSV⟩.

CHEN, D. et al. Coherent online video style transfer. In *2017 IEEE International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2017. p. 1114–1123. ISSN 2380-7504.

CHENG, Z.; YANG, Q.; SHENG, B. Deep colorization. *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, p. 415–423, 2015. ISSN 15505499.

CIRESAN, D. C. et al. Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*. [S.l.]: AAAI Press, 2011. p. 1237–1242. ISBN 978-1-57735-514-4.

DENG, J. et al. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*. [S.l.: s.n.], 2009.

GREFF, K. et al. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, p. 2222–2232, 2017. ISSN 2162-237X.

HUANG, Y.-C. et al. An adaptive edge detection based colorization algorithm and its applications. In *Proceedings of the 13th Annual ACM International Conference on Multimedia*. New York, NY, USA: ACM, 2005. (MULTIMEDIA '05), p. 351–354. ISBN 1-59593-044-2.

HUBERT, I. *Tears of Steel: an open-source movie*. 2018. Available from Internet: ⟨https://mango.blender.org/⟩.

IIZUKA, S.; SIMO-SERRA, E.; ISHIKAWA, H. Let there be color! *ACM Transactions on Graphics*, vol. 35, no. 4, p. 1–11, 2016. ISSN 07300301. Available from Internet: ⟨http://dl.acm.org/citation.cfm?doid=2897824.2925974⟩.

KERAS. *Keras Surgeon: package for weight pruning*. 2018. Available from Internet: ⟨https://github.com/BenWhetton/keras-surgeon⟩.

KUMAR, R. et al. A learning-based approach for automatic image and video colorization. *Computer Graphics International, Bournemouth, UK.*, no. June, 2012.

LABS, I. *Bringing Parallelism to the Web with River Trail*. 2018. Available from Internet: ⟨http://intellabs.github.io/RiverTrail/tutorial/⟩.

LARSSON, G.; MAIRE, M.; SHAKHNAROVICH, G. Learning representations for automatic colorization. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9908 LNCS, p. 577–593, 2016. ISSN 16113349.

LEVIN, A.; LISCHINSKI, D.; WEISS, Y. Colorization using optimization. *ACM SIGGRAPH 2004 Papers on - SIGGRAPH '04*, p. 689, 2004. ISSN 07300301.

MATIAS, R.; JORGE, V.; JAVIER, B. Image Colorization with Neural Networks. no. i, p. 1–4, 2016.

MOCHA. *Tracking Software Mocha*. 2018. Available from Internet: ⟨https://borisfx.com/products/mocha/⟩.

PANBOONYUEN, T. *Road Segmentation of Remotely-Sensed Images Using Deep Convolutional Neural Networks with Landscape Metrics and Conditional Random Fields*. 2018. Available from Internet: ⟨https://www.researchgate.net/figure/A-proposed-network-architecture-for-object-segmentation-exponential-linear-unit_fig2_318125611⟩.

PAPERSPACE. *Paperspace Cloud Provider*. 2018. Available from Internet: ⟨https://www.paperspace.com/⟩.

REVAUD, J. et al. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2015. p. 1164–1172. ISSN 1063-6919.

RUDER, M.; DOSOVITSKIY, A.; BROX, T. Artistic style transfer for videos. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9796 LNCS, p. 26–36, 2016. ISSN 16113349.

SHIBATA, T. K. C.; BARROS, H. C. S.; TANAKA, V. A. *Dataset for the ColorMotion project*. 2018. Available from Internet: ⟨https://github.com/ColorMotion/Dataset⟩.

SHIBATA, T. K. C.; BARROS, H. C. S.; TANAKA, V. A. *Docker for the ColorMotion project*. 2018. Available from Internet: ⟨https://github.com/ColorMotion/ColorMotion-docker⟩.

SILHOUTTE. *Tracking Software Silhoutte*. 2018. Available from Internet: ⟨https://www.silhouettefx.com/⟩.

TKALCIC, M.; TASIC, J. F. Colour spaces: perceptual, historical and applicational background. In *The IEEE Region 8 EUROCON 2003. Computer as a Tool.* [S.l.: s.n.], 2003. vol. 1, p. 304–308 vol.1.

WEINZAEPFEL, P. et al. Deepflow: Large displacement optical flow with deep matching. In *Proceedings of the 2013 IEEE International Conference on Computer Vision*. [S.l.]: IEEE Computer Society, 2013. (ICCV '13), p. 1385–1392. ISBN 978-1-4799-2840-8.

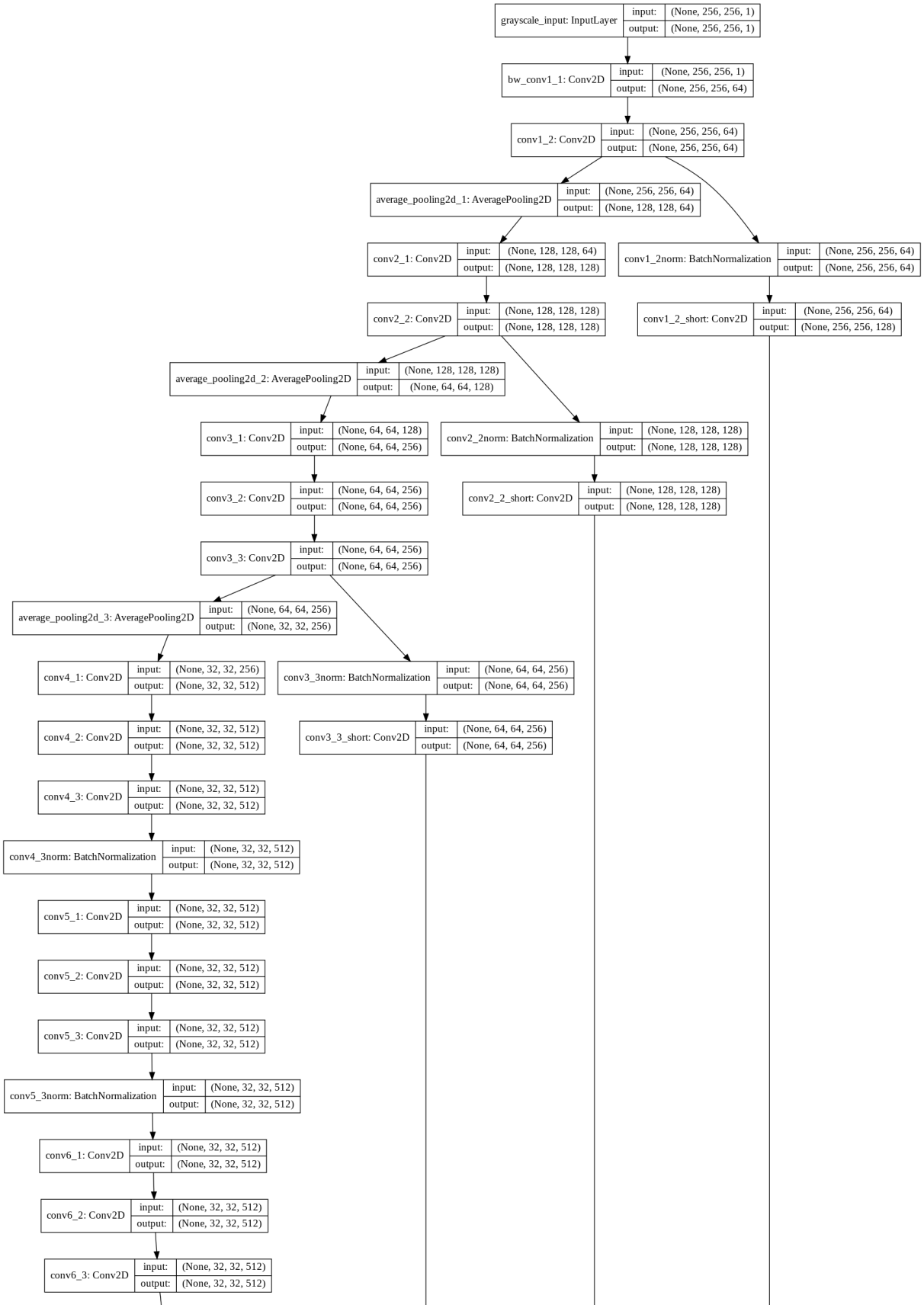
YAO, J. *Repository for the MMdnn project*. 2018. Available from Internet: ⟨https://github.com/Microsoft/MMdnn⟩.

ZHANG, R.; ISOLA, P.; EFROS, A. A. Colorful image colorization. In SPRINGER. *European Conference on Computer Vision*. [S.l.], 2016. p. 649–666.

ZHANG, R. et al. Real-Time User-Guided Image Colorization with Learned Deep Priors. 2017. ISSN 0730-0301. Available from Internet: ⟨http://arxiv.org/abs/1705.02999⟩.

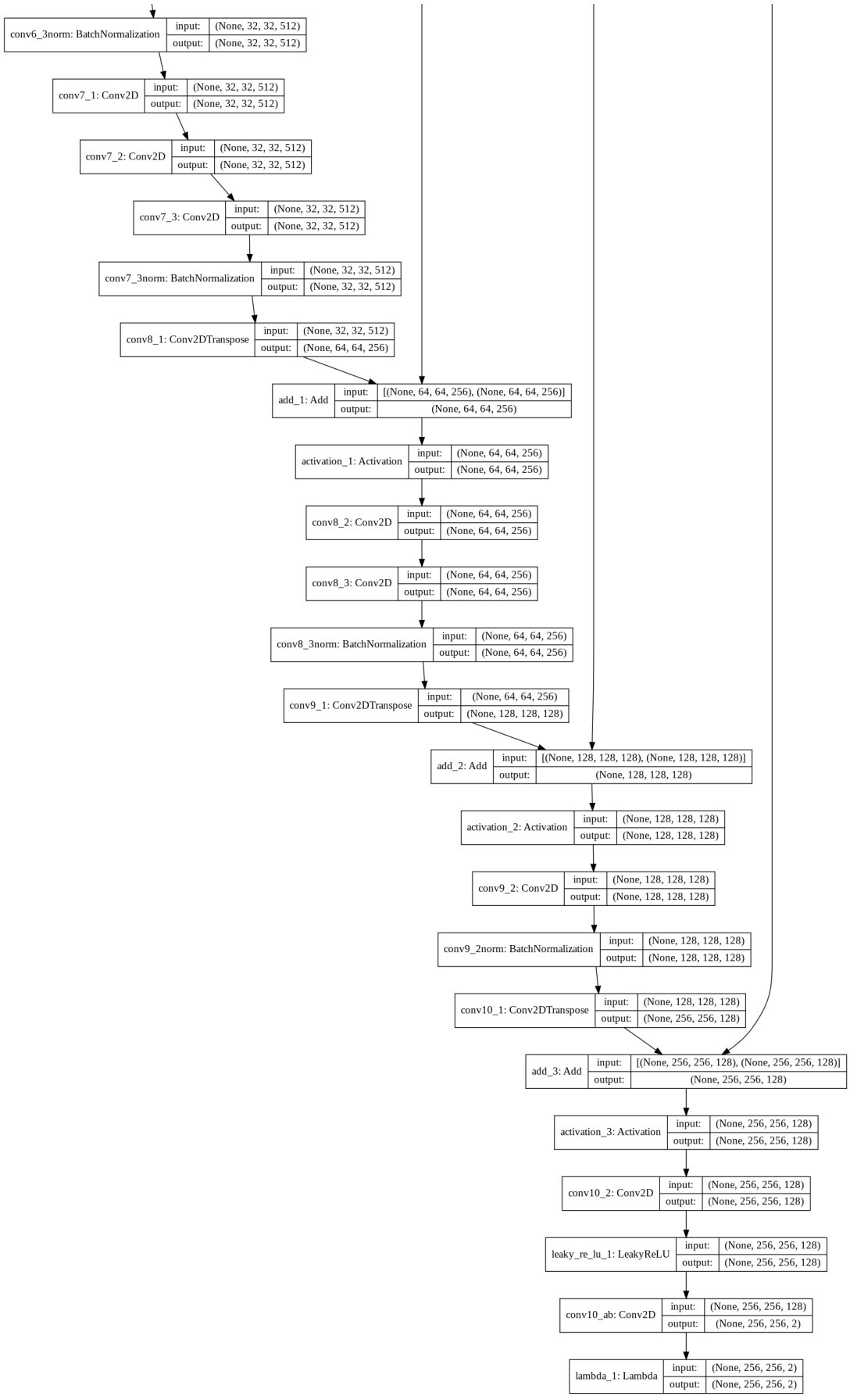
APPENDIX A – Simple architecture model

Figure 15 – Simple model architecture expanded - Part 1.



Source: authors.

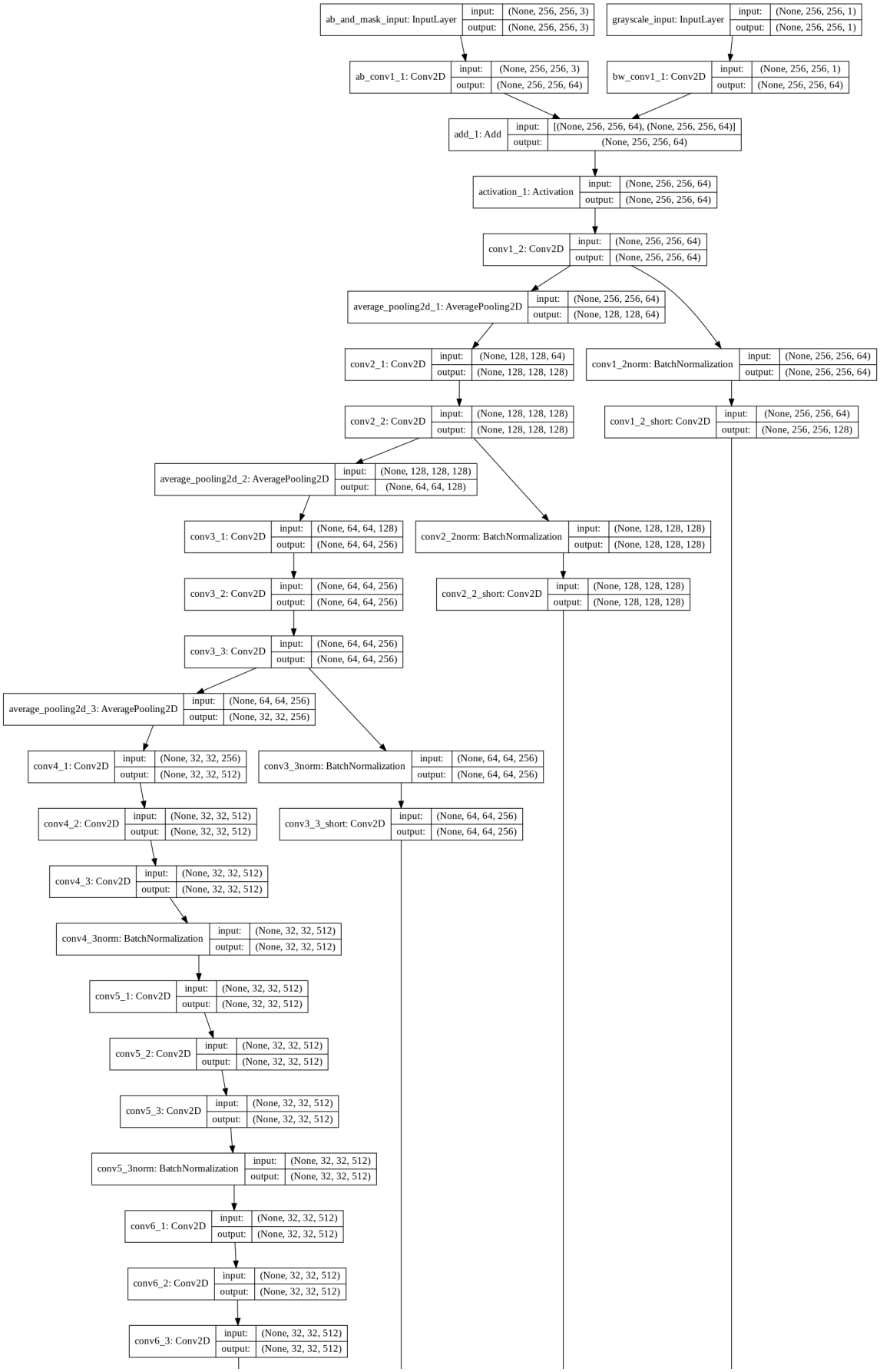
Figure 16 – Simple model architecture expanded - Part 2.



Source: authors.

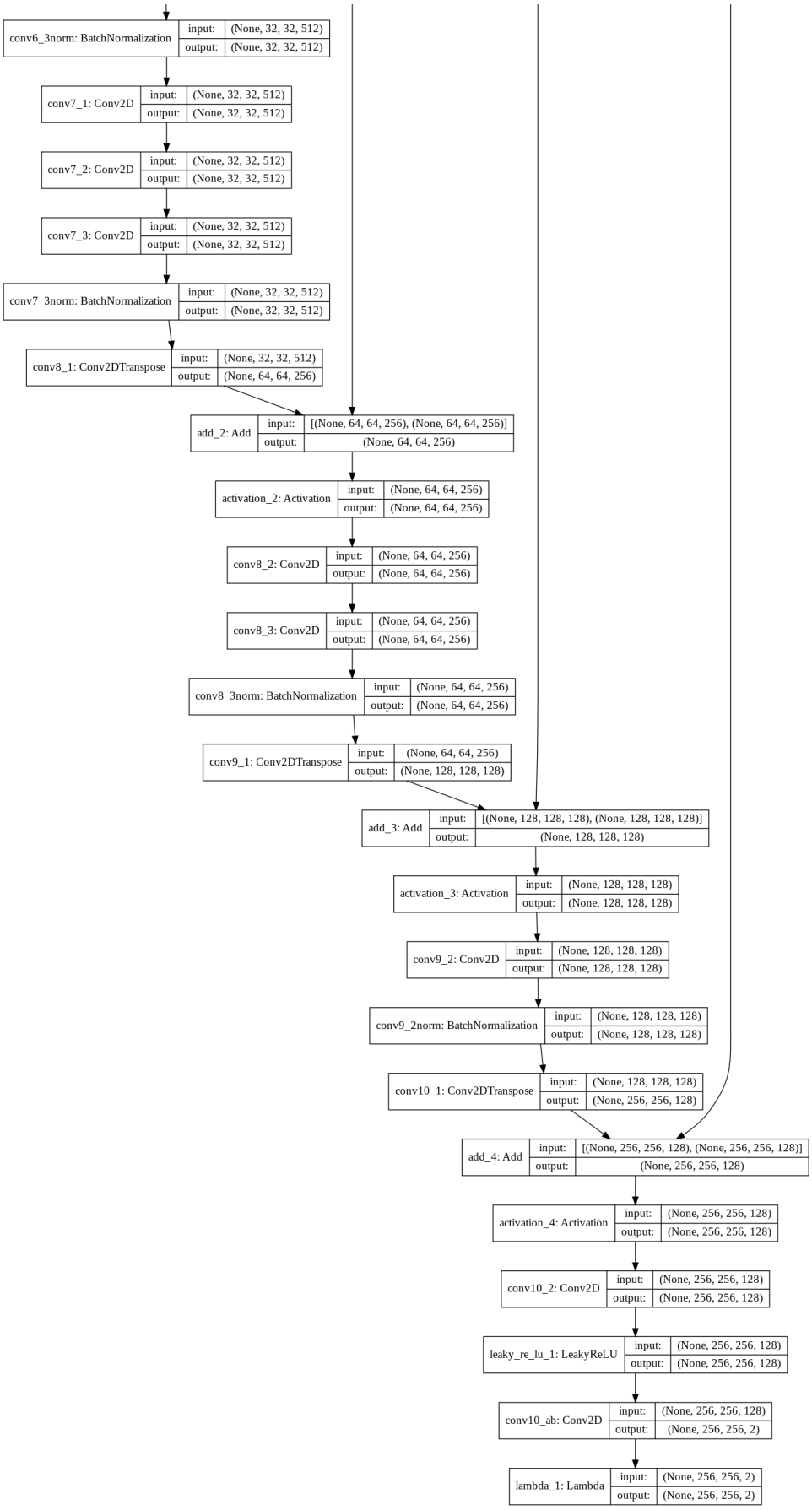
APPENDIX B – Guided architecture model

Figure 17 – Guided model architecture expanded - Part 1.



Source: authors.

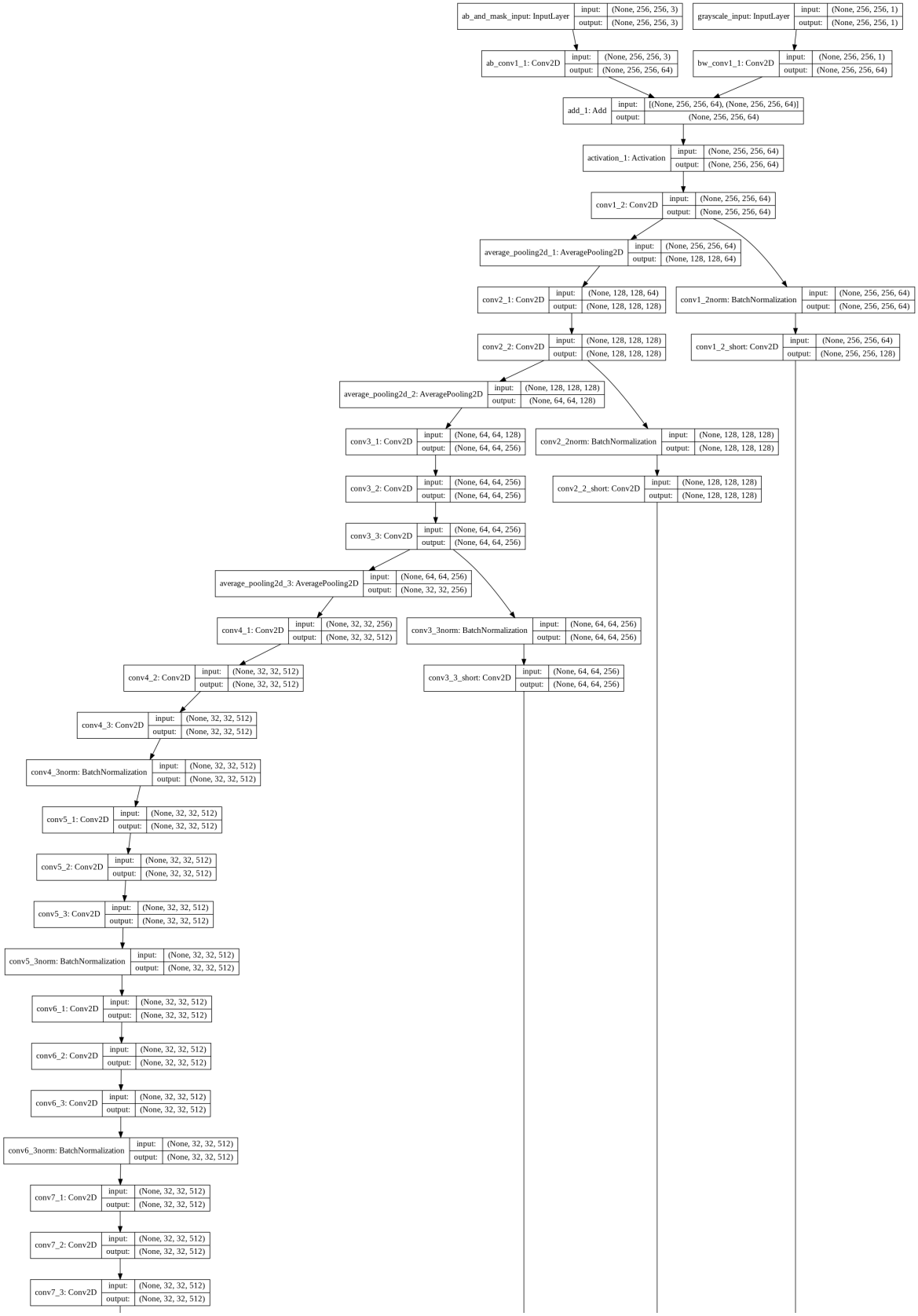
Figure 18 – Guided model architecture expanded - Part 2.



Source: authors.

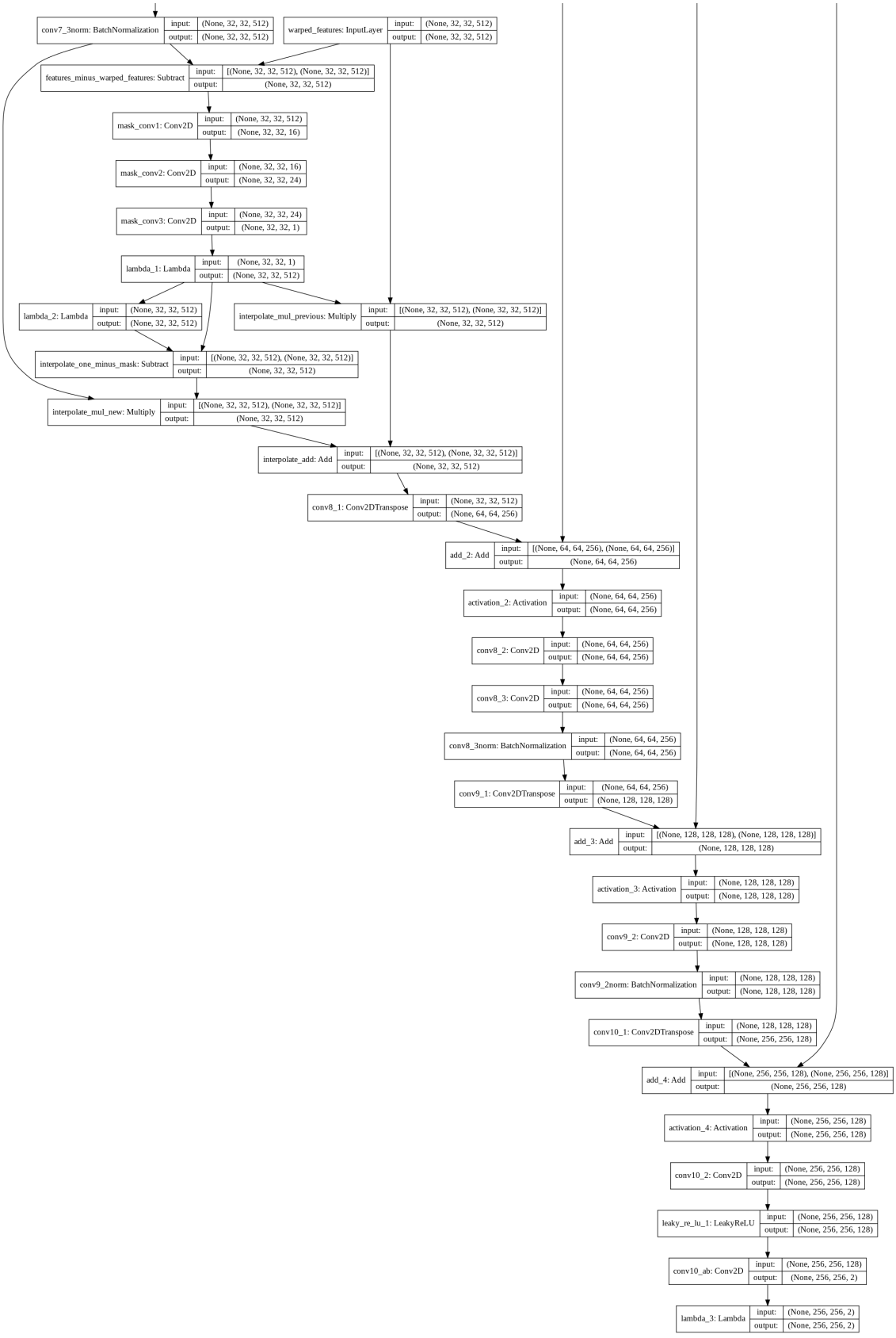
APPENDIX C – Stateful architecture model

Figure 19 – Stateful model architecture expanded - Part 1.



Source: authors.

Figure 20 – Stateful model architecture expanded - Part 2.



Source: authors.