Deep Convolutional Neural Network for Non-rigid Image Registration

Eduard F. Durech
Simon Fraser University, Burnaby, Canada
Email: EDurech@SFU.ca

Abstract—Images taken at different times or positions undergo transformations such as rotation, scaling, skewing, and more. The process of aligning different images which have undergone transformations can be done via registration. Registration is desirable when analyzing time-series data for tracking, averaging, or differential diagnoses of diseases. Efficient registration methods exist for rigid (including linear or affine) transformations; however, for non-rigid (also known as non-affine) transformations, current methods are computationally expensive and time-consuming. In this report, I will explore the ability of a deep neural network (DNN) and, more specifically, a deep convolutional neural network (CNN) to efficiently perform non-rigid image registration. The experimental results show that a CNN can be used for efficient non-rigid image registration and in significantly less computational time than a conventional Diffeomorphic Demons or Pyramiding approach.

I INTRODUCTION

Morphing of images due to transformations introduced by time, non-idealistic properties, or multi-modal acquisitions make direct comparison of images challenging. Examples of influences leading to transformations can be movement of the acquisition device, environmental influences such as temperature, or aberrations. In medical imaging, time-series data are independently transformed and some subject images are entirely different from a template image such as in Magnetic Resonance Imaging (MRI) [1]. The process of aligning images, either different images or the same image under different transformations, can be achieved via registration [2, 3].

Algorithmic registration methods exist but are time-consuming and computationally expensive for more complicated registration tasks. Deep Learning has shown success in similar image processing problems such as image segmentation using a deep convolutional neural network (CNN) [4]. CNN’s can be extended to develop image transformation maps, in much the same way they develop segmentation maps, for a subject image in order to transform and align it with a template.

This paper will explore using a CNN for developing the transformation maps to register medical images with emphasis on axial brain MRI scans. The purpose is to develop an efficient method for registering images which can greatly contribute to medical image quality, processing, and diagnostics.

A Registration

Registration is the act of aligning structures within differing images. Differences may arise due to different subjects and/or different image transformations. In medical image registration, anatomical structures are aligned leading to the ability to compare against standard templates or for analyzing the impacts of disease progression over time. Beyond the direct diagnostic and clinical significance of image registration, successful and efficient registration has the ability to streamline image processing pipelines and enhance the quality of medical images by taking multiple images in series and averaging them [5].

Image registration relies on the assumption that transformations are independently capable of morphing an image to align with another. This assumption holds true for most medical imaging but is unable to resolve overlapping of parts in an image if information of the hidden part is unavailable, such as an anatomical part of the body being hidden behind another. This limits classic registration techniques to aligning available information in a subject image, $S$, to available information in a template image, $T$.

The process of deriving the correct transformation to align $S$ and $T$ is considered the process of image registration. Once the transformation is applied to $S$, the resulting image is said to be registered to $T$. Transformations, and the process of deriving them for registration, can be grouped into two forms, rigid and non-rigid.

1 Rigid Registration

Aligning images using linear transformations belongs to the class of rigid registration. Rigid registration treats the transformation as a rigid body which must preserve Eu-
clidean distance between points. Examples of rigid transformations includes rotations and/or translations. Affine transformations are an extension of rigid transformations which preserve lines and parallelisms. Examples of affine transformations include scaling, shears, reflections, or any combination of them.

A plethora of methods exist for deriving rigid transformations used in rigid registration. A popular example for translation is phase correlation which uses peaks found in the cross-power spectrum of images. An efficient method for computing phase correlation is given by \[ \theta \]

\[ \arg \max \left\{ \mathcal{F}^{-1} \left\{ \frac{S \circ T^*}{|S \circ T^*|} \right\} \right\} \]

with \( i, j \) representing the \( i \)th row and \( j \)th column of an image, \( \mathcal{F} \) the 2D Fourier transform, \( S = \mathcal{F}\{s\} \) the 2D Fourier transform of the subject image, \( T = \mathcal{F}\{t\}^* \) the conjugate of the 2D Fourier transform of the template image, and \( \circ \) the Hadamard product. This method easily extends to the \( n \)th dimension by replacing \( i, j \) with \( n \) arguments and taking the \( n \)D Fourier transform in place of 2D. Phase correlation can further be extended to derive the scale and rotation transformation by initially converting \( S \) and \( T \) to log-polar.

2 Non-rigid Registration

As rigid and affine transformations are, respectively, limited to preserving Euclidean distances between points or lines and parallelisms, they falter when attempting to register images which have undergone non-rigid transformations. Non-rigid, also known as non-affine or elastic, transformations can be thought of as local rigid or affine transformation. Non-rigid transformations do not need to preserve any qualities and can independently shift each pixel in an image.

Difficulties arise in attempting to derive non-rigid transformations for registration as the inherent limitless characteristics of these transformations means parts may overlap or follow non-linear properties. Classic methods rely on pyramiding and resolving registration for multiple resolutions such as Diffeomorphic Demons \cite{7, 8}. These methods are robust but require a lot of computational time to complete as well as parameter tuning for specific cases. Deep learning methods for 3D CT registration have shown promising results in predicting the needed transformations for registration \cite{9, 10}; however, these methods were developed for 3D and feature large architectures which increase computational time.

B Warp Fields

Non-rigid transformations can be represented by the shifts experienced by an individual or group of pixels, known as warp fields or maps. One way warp fields can be visualized is by taking a square grid and applying the transformation to it, as seen in Fig. 1. Mathematically, warp fields, \( \varphi \), can be applied to images to attain a warped image, \( W \), as

\[ W[i, j] = S[i, j] \circ \varphi[i, j] = S[i - \varphi[i, j], j - \varphi[j][i, j]]. \]

As some pixels will be left empty after non-rigid transformations, a common practice is to fill in empty pixels using interpolation.

C Image Similarity

Quantifying similarity between images is not trivial and no universal method exists. Registration techniques generally classify similarity by intensity or features. Pyramiding commonly uses normalized intensities and finds the greatest cross-correlation at each step. This method, however, is prone to "over-registering" by aligning parts of an image whose pixels may have the same intensity but would not hold faithful to the structure of the image. An example would be if a hot-pixel were located in one item of the subject image but in another item of the template image; an ideal intensity-based registration would move the hot-pixel into the other item even if it did not belong there. Feature-based registration is possible but requires development of feature quantification algorithms, of which no universal method exists.

1 Mean Squared Error (MSE)

Image intensity optimizations generally minimize the mean squared error (MSE) between the warped image and template image. The error, MSE, is computed for every pixel in the image as given by

\[ \text{MSE} = \frac{1}{I \cdot J} \sum_{i,j} (W[i, j] - T[i, j])^2. \]

2 Structural Similarity Index Measure (SSIM)

One way to quantify feature similarity in images is by using their structural similarity. A popular method known as the structural similarity index measure (SSIM) computes
the structural similarity between two windows, \( x \) and \( y \), as

\[
\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)},
\]

where \( \mu_{x,y} \), \( \sigma^2_{x,y} \) represent the mean and variance of \( x \) and \( y \), respectively, \( \sigma_{xy} \) represents the covariance of \( x \) and \( y \), and \( c_{1,2} \) are parameters usually defined as the weighted dynamic range of pixel values.

SSIM has an advantage over MSE in defining how "similar" images are. As MSE cannot distinguish between more than intensity at a given pixel, it can give the same error for source images which are both similar and very different from a template image. An example of this is displayed in Fig. 2 which shows a hypersphere of different images around a reference image of equal-MSE. It can be seen that images of varying similarity to the reference image can have the same MSE, but SSIM can distinguish structural similarities and properly assigns a high SSIM for similar images and a low SSIM for differing ones.

**D Autoencoders**

Autoencoders are a type of deep neural network (DNN) which encode a higher-dimensional input into a lower-dimensional latent space and then decode them back to a higher-dimension. Their supporting theory is given by the encoder filtering redundant information and representing the input in the lower-dimensional latent space which contains the most critical information about the data. The decoder will then use this critical information to generate a higher-dimensional representation of the data.

Convolutional autoencoders are common to use for deep learning-based approaches to image processing such as image segmentation. One well-known example is U-Net \[4\] which uses an autoencoder structure while concatenating the outputs of the encoding levels with their respective decoding levels. The same method can be used for generating warp fields for registration instead of segmentation maps \[9, 10\].

**II Methods**

The overall pipeline of the proposed method includes inputting the subject image, \( S \), and template image, \( T \), which are then processed and fed into a DNN. The DNN creates the warp fields, or transformation maps, \( \varphi_j \) and \( \varphi_i \), and warps \( S \) to obtain the warped image, \( W \). The warp fields were chosen to be 2D representations of the shifts for each pixel in their respective dimension, \( j \) for column or \( i \) for row. The pipeline is visualized in Fig. 3.

Images were normalized before being inputted into the DNN. The input data was defined as the normalized source image and normalized template images, shown by example in Fig. 4. The choice to use simply the source and template image was to not make any assumptions of how the DNN would behave. Methods such as BIRNet \[9\] guide the DNN by inputting the difference between the subject and template as well as a Sobel-filtered subject image along with the subject image. While making assumptions about model behaviours can lead to faster training, it can also confound and restrict the ability of the model to learn. As such, minimizing the amount of information fed into a DNN can be beneficial and poses a lower risk of confusing the model.

**Fig. 3:** General pipeline.

**Fig. 4:** Example DNN input showing the normalized source image and template.
A Deep Convolutional Neural Network (CNN)

As convolutional autoencoders are known to be good architectures for generative image-processing tasks, a modified version of U-Net [4] was used as the DNN. The main modifications are a change in dimensionality of the filters to distinguish higher-resolution differences in images, a 2-dimensional input of $S_{i,j}$ and $T_{i,j}$, a 2-dimensional output for $\varphi_j$ and $\varphi_i$, an additional warping step of the warp fields with $S_{i,j}$, and a novel loss function. Hyperparameters were tuned to a learning rate of $10^{-4}$ for Adam optimizer with a batch size of 1 trained for 100 epochs. All hidden layers used a tanh activation while the final layer used a linear activation to preserve the negative and positive values of the warp field.

The loss was computed between $W$ and $T$. As MSE does not capture the structural similarity between images, it was not used as the primary loss. SSIM could be used independently, but as there is no universal method of quantifying feature-based losses, a weighted addition of the two was used. As MSE should be minimized and SSIM maximized (maximal at 1), and DNN’s generally seek to minimize loss, the loss was quantified as

$$\text{Loss} = \alpha \cdot \text{MSE} - \beta \cdot (\text{SSIM} - 1),$$

which, in an ideal scenario with two exact images, would result in a loss of 0. The weighting coefficients $\alpha$ and $\beta$ were set to 10 and 1, respectively. These values were chosen to make the losses of similar magnitude for this application so both have similar influence on the loss.

Loss was not computed between warp fields as images, such as brain MRI scans, can contain many empty pixels which would be invariant to some transformations. In this case, calculating the loss using warp fields could confuse the network into learning undesirable warp fields. This also increases the variability of solutions and thus complexity in convergence. An example of this would be rotating the outside empty region of every image with little effect on loss. As aligning the warped and template images is the ultimate goal of the network, the loss was only calculated between them.

B Data Preparation

200 axial brain MRI slices were obtained from a TCIA public dataset [15] and down-sampled to 128x128 pixels. Each image underwent 5 random warps defined by linear, spherical, and sinusoidal warp fields in $i$ and $j$. The subsequent source image and template image (before warping) were used as the input. The warped image warped once more by its inverse field was used as the ground truth. A training/validation split of .8/.2 was used.

III Experiments and Results

To test the impact of training losses on the final outcome of the model, three losses were logged for their training and validation accuracy: MSE-SSIM, SSIM (minimized as 1-SSIM), and MSE. The effects of training progression were also explored to see if the network was improving its registration over time and not just minimizing its loss.

A Losses

A comparison of the three different training losses can be seen in Fig. 6. SSIM outperforms MSE in training and validation when monitoring SSIM, as is expected; however, SSIM also minimizes validation MSE greater than training with a standalone MSE loss, and is similar in training loss. This may be due to SSIM being better-suited for the problem of registration, as previously discussed. The proposed MSE-SSIM loss outperforms both a standalone SSIM or MSE loss in minimizing both validation SSIM and MSE.

B Training Progression

As training progressed, the source image became more similar to the template image as shown in Fig. 7. Quantitatively, SSIM increased and MSE decreased with increasing epochs, as desired; qualitatively, the false-colour overlay shows less green and magenta and more grey, meaning the images are better-aligned after longer training. This shows the network learned some representation of non-rigid image registration.
IV EXPERIMENTAL RESULTS

This method effectively generates non-rigid warp fields to register a source image to a template as can be seen in Fig. 8. In the example cases, SSIM was roughly doubled while MSE was reduced by an order of magnitude. This method also efficiently generates the warp field and warps the image with an average compute time of .034s per 128x128 image taken over 100 runs.

While a lack of ground truth exists for real-world non-rigid warp fields, the synthetic data used to create these examples could be used for quantifying accuracy; however, this method of analysis suffers the same shortfalls as using the warp fields in the loss. As previously discussed, warp fields in certain regions, such as regions with empty pixels, may not impact the structural transformation of the image. Registering or calculating accuracy including these redundant regions does not give a faithful representation of the model’s ability.

The warped output image, which is the desired point of reference for how well the model is performing, should be used for comparisons. A shortcoming of using the warped image is the constraint of how well we can quantify similarity between the warped and template images, as with losses. SSIM and MSE will both be used for comparisons by convention and because there is some correlation between these metrics and how well the images are aligned.

A Comparison with Algorithmic Methods

Computational time and metric comparisons with a conventional pyramiding approach to non-rigid image registration using Diffeomorphic Demons [7, 8] were averaged over 100 samples. The Diffeomorphic Demons algorithm was swept through an increasing amount of iterations and levels for comparison with the DNN.

Diffeomorphic Demons showed superior SSIM and MSE metrics under certain conditions, but took significantly longer time to compute the transformations compared to the DNN, as seen in Fig. 9. The DNN’s metrics are as low as 15% less than a fine-tuned Diffeomorphic Demons algorithm, but the DNN is computationally faster by at least an order of magnitude in comparison with the fastest Diffeomorphic Demons configuration.

B Probing the Decision-Making Process

In order to understand the decision-making process of the DNN, a test image can be run through the trained network and intermediate outputs from each convolutional level can be shown.

1 Convolutional Filters

Outputs of various convolutional levels in both the encoder and decoder can be seen for a test image in Fig. 10. The initial outputs of the encoding levels resemble that of edge detectors which retain structure but dilute the amount of information into a latent space. The decoding level outputs begin to generate images which no longer represent the input, but instead that of a warp field. As the decoder uses deconvolutions, grid patterns arise while expanding the latent space.

It can be seen in Fig. 10 at the 256 output-level that the network seems to be focusing on specific sub-regions to
Fig. 8: Example results showing comparisons of inputs to outputs by SSIM, MSE, and a false-colour map of each in green versus the ground truth in magenta (grey means there is overlap). Below each example are the DNN-generated warp fields.
transform which become even further generalized in the 128 output-block. This is analogous to the behaviour of pyramid- ing by windows of varying resolutions. The network can be interpreted as a "smart" image processor which selectively weights certain windows and regions of an image after being filtered and encoded into a constrained latent-space. This constrained latent-space features critical information such as edges and structure used in the decoding levels.

2 Output Filter

The final convolutional output layer before creating the warp field shows sub-parts used to derive the transformation. Fig. 11 displays all outputs from this layer for a test image. Interesting patterns arise which seem to selectively isolate regions of the 2D image. These isolated regions are convolved once more to derive the final two warp maps. Some patterns show structures and edges likely used for aligning hard structure while the majority of outputs appear to be rotational isolations of the image. These likely represent sub-regions of spherical warp fields.

Fig. 9: Comparison of DNN with Diffeomorphic Demons parameter sweeps for SSIM (top) and MSE (bottom).
Note: x-axes are in log-scale.

Fig. 11: All final filtered-outputs for a test image.
V Discussion

Although a CNN approach is an order of magnitude faster than an algorithmic approach, the quality of the registration may not be sufficient for certain use-cases. In averaging of multiple images, source images should be near-perfectly aligned or else blurring will occur. This method has the capability of improving with different architectures, loss functions, hyperparameter tuning, and better training data. For applications which can suffice with an SSIM of <.85 and time is of concern, this method proves a good alternative to the fastest versions of algorithmic methods and boasts a higher SSIM in considerably less time. This can prove especially important for larger data as the computational demands of the algorithmic methods are non-linear for the same registration resolution.

To better represent the warp fields, \( \varphi_{i,j} \) can be reparametrized in polar coordinates as \( \varphi_{\rho, \theta} \). This could help resolve issues that may arise when attempting to register images which have transformations beyond the convolutional windows. Although it wasn’t tested, such transformations could prove challenging to resolve using square convolution kernels of set dimensions. Another potential alleviation to this problem is circular convolutions.

VI Conclusion

Non-rigid image registration can be efficiently attained by DNN’s and, more specifically, CNN’s. They can achieve better quality versus time than algorithmic methods such as Diffeomorphic Demons. The proposed method uses a convolutional autoencoder to produce warp fields given a source and template image. The warp fields non-rigidly transform the source image to align with the template image, attaining registration. The network was guided using a weighted MSE-SSIM loss which achieved better SSIM and MSE metrics than either as a standalone loss. The network was also probed and its decision-making process visualized by means of displaying intermediate results throughout the network using a test image. It was discovered that the convolutional filters behave in a similar manner as algorithmic approaches do by pyramiding resolutions and windows for refined and gross transformations. This method can still be developed in order to improve its metrics and generality.

VII Acknowledgment

I would like to thank my Professor, Dr. Ivan Bajic, for the knowledge given about deep learning systems in ENSC 413.

References

[1] I. N. Bankman, Handbook of Medical Imaging: Processing and Analysis Management. Academic Press, 2000.
[2] J.B. Antoine Maintz and M. A. Viergever, “A survey of medical image registration,” Medical Image Analysis, vol. 2, 1998.
[3] A. Klein, J. Andersson, B. Ardekanl, J. Ashburner, B. Avants, M. C. Chiang, G. E. Christensen, D. L. Collins, J. Gee, P. Hellier, J. H. Song, M. Jenkinson, C. Lepage, D. Rueckert, P. Thompson, T. Vercauteren, R. P. Woods, J. J. Mann, and R. V. Parsey, “Evaluation of 14 nonlinear deformation algorithms applied to human brain MRI registration,” NeuroImage, vol. 46, no. 3, pp. 786–802, 2009.
[4] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in Medical Image Computing and Computer-Assisted Intervention (MICCAI), ser. LNCS, vol. 9351, 2015, pp. 234–241.
[5] A. Uji, S. Balasubramanian, J. Lei, E. Baghdasaryan, M. Al-Sheikh, and S. R. Sadda, “Impact of multiple en face image averaging on quantitative assessment from optical coherence tomography angiography images,” Ophthalmology, vol. 124, no. 7, pp. 944–952, 2017.
[6] C. D. Kuglin and D. C. Hines, “The phase correlation image alignment method,” Proceeding of IEEE International Conference on Cybernetics and Society, pp. 163–165, 1975.
[7] J. Thirion, “Image matching as a diffusion process: an analogy with maxwell’s demons,” Medical Image Analysis, vol. 2, no. 3, pp. 243–260, 1998.
[8] T. Vercauteren, X. Pennec, A. Perchant, and N. Ayache, “Diffeomorphic demons: Efficient non-parametric image registration,” NeuroImage, vol. 45, no. 1, pp. S61–S72, 2009.
[9] J. Fan, X. Cao, P. T. Yap, and D. Shen, “BIRNet: Brain image registration using dual-supervised fully convolutional networks,” 2018, arXiv:1802.04692.
[10] H. Sokooti, B. de V os, F. Berendsen, M. Ghafoorian, S. Yousefi, B. P. F. Lelieveldt, I. Igum, and M. Staring, “3D convolutional neural networks image registration based on efficient supervised learning from artificial deformations,” 2019, arXiv:1908.10235.
[11] A. Myronenko and X. Song, “Adaptive regularization of ill-posed problems: Application to non-rigid image registration,” 2009, arXiv:0906.3323.
[12] Z. Wang, E. P. Simoncelli, and A. C. Bovik, “Multiscale structural similarity for image quality assessment,” in The Thirty-Seventh Asilomar Conference on Signals, Systems Computers, 2003, vol. 2, 2003, pp. 1398–1402 Vol.2.
[13] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600–612, 2004.
[14] ——, “The ssim index for image quality assessment,” 2014. [Online]. Available: https://ece.uwaterloo.ca/~z70wang/research/ssim/
[15] TCI, “The cancer genome atlas low grade glioma (tcga-lgg),” 2020. [Online]. Available: https://wiki.cancerimagingarchive.net/display/Public/TCGA-LGG
VIII APPENDIX A TENSORFLOW PYTHON LOSSES

```python
1  def SSIM_loss(T, S):
2     return tf.reduce_mean(tf.image.ssim(T, S, 1.0))
3
4  def MSE_SSIM(T, S, alpha=10, beta=1):
5     return alpha*MSE(T, S) - beta*(SSIM_loss(T, S)-1)
```

IX APPENDIX B KERAS PYTHON CNN ARCHITECTURE

```python
1  ModelIn = Input((128,128,2))
2
3  S = Lambda(lambda x: x[:,:,:,0:1])(ModelIn)
4
5  # Left Layer 1
6  c1 = Conv2D(64, 3, activation='tanh', padding='same')(ModelIn)
7  c2 = Conv2D(64, 3, activation='tanh', padding='same')(c1)
8  dr1 = Dropout(0.5)(c2)
9
10  # Left Layer 2
11  p1 = MaxPooling2D((2, 2))(dr1)
12  c3 = Conv2D(128, 3, activation='tanh', padding='same')(p1)
13  c4 = Conv2D(128, 3, activation='tanh', padding='same')(c3)
14  dr2 = Dropout(0.5)(c4)
15
16  # Left Layer 3
17  p2 = MaxPooling2D((2, 2))(dr2)
18  c5 = Conv2D(256, 3, activation='tanh', padding='same')(p2)
19  c6 = Conv2D(256, 3, activation='tanh', padding='same')(c5)
20  dr3 = Dropout(0.5)(c6)
21
22  # Left Layer 4
23  p3 = MaxPooling2D((2, 2))(dr3)
24  c7 = Conv2D(512, 3, activation='tanh', padding='same')(p3)
25  c8 = Conv2D(512, 3, activation='tanh', padding='same')(c7)
26  dr4 = Dropout(0.5)(c8)
27
28  # Bottom Layer
29  p4 = MaxPooling2D((2, 2))(dr4)
30  c9 = Conv2D(1024, 3, activation='tanh', padding='same')(p4)
31  c10 = Conv2D(1024, 3, activation='tanh', padding='same')(c9)
32
33  # Right Layer 4
34  uc4 = Conv2DTranspose(512, (2, 2), padding='same', strides=(2,2))(c10)
35  uc4 = concatenate([uc4,c8], axis=-1)
36  cu8 = Conv2D(512, 3, activation='tanh', padding='same')(uc4)
37  cu7 = Conv2D(512, 3, activation='tanh', padding='same')(cu8)
38
39  # Right Layer 3
40  uc3 = Conv2DTranspose(256, (2, 2), padding='same', strides=(2, 2))(cu7)
41  uc3 = concatenate([uc3,c6], axis=-1)
```
42 cu6 = Conv2D(256, 3, activation='tanh', padding='same')(uc3)
43 cu5 = Conv2D(256, 3, activation='tanh', padding='same')(cu6)

44 # Right Layer 2
45 uc2 = Conv2DTranspose(128, (2, 2), padding='same', strides=(2, 2))(cu5)
46 uc2 = concatenate([uc2, c4], axis=-1)
47 cu4 = Conv2D(128, 3, activation='tanh', padding='same')(uc2)
48 cu3 = Conv2D(128, 3, activation='tanh', padding='same')(cu4)

50 # Right Layer 1
51 uc1 = Conv2DTranspose(64, (2, 2), padding='same', strides=(2, 2))(cu3)
52 uc1 = concatenate([uc1, c2], axis=-1)
53 cu2 = Conv2D(64, 3, activation='tanh', padding='same')(uc1)
54 cu1 = Conv2D(64, 3, activation='tanh', padding='same')(cu2)

56 Phi = Conv2D(2, (1, 1), activation='linear', padding='same')(cu1)
57 W = Lambda(lambda args: tf.contrib.image.dense_image_warp(*args))([S, Phi])

60 FullModel = Model(inputs=ModelIn, outputs=W)
61 FullModel.compile(optimizer=Adam(lr=1e-4), loss='MSE_SSIM')
62 FullModel.fit(np.concatenate((x, y), axis=-1), y, batch_size=1, epochs=100, \\
64   validation_split=0.2)