Image Similarity Models with the TotallyLooksAlike Dataset

Zimo Peng
Masters of Information Technology
The University of Melbourne
Melbourne, Australia
zimop@student.unimelb.edu.au

Brandon Widjaja
Masters of Information Technology
The University of Melbourne
Melbourne, Australia
bwwid@student.unimelb.edu.au

***Abstract*- Matching similar images within large datasets has been a heavily researched area of computer vision for many years, with many robust models already developed. We attempt to solve this problem by 2 models, the convolutional autoencoder, and the Siamese network.**

I. INTRODUCTION

Image similarity stands as a fundamental challenge in the field of computer vision. In this report, we discuss the implementation of convolutional (CNN) autoencoders and Siamese networks, as well as the potential of different strategies such as transfer learning and different loss functions for the purpose of designing a model that can identify similar images from a batch of random images. This report aims to explore the methodologies we implemented to attempt to solve this problem, as well as to critically analyse the performance of these models and the impacts of any changes we experimented with. We also aim to show the strengths and weaknesses of each approach used during this challenge.

II. METHODOLOGIES

2.1 Autoencoder

The first option we went with was the CNN autoencoder for image similarity. Autoencoders are neural networks, designed to encode an input into a compressed and meaningful representation. This is then decoded back to reconstruct an input as similar as possible to the original one [1]. The key idea is that the neural network is forced to learn the most salient features of an input, creating a compressed feature embedding.

For our task, we have adapted the idea of image denoising using autoencoders, by creating a CNN autoencoder (an autoencoder with convolutional layers). Which is fed a right image from the encoder side, and having the decoder attempt to output a left image, to learn the feature embeddings of what makes 2 images similar to each other.

To prepare our data, we resize the images to 128x128 , so that each down sample output is “even” for at least 3 hidden layers..

2.1.1 CNN Architecture

Our CNN autoencoder consists of 6 hidden layers. The encoder has 3 Conv2D hidden layers with 32 filters with a 3x3 kernel. Each layer is followed by a max pooling layer to reduce complexity and focus on important features, and each layer uses the LeakyRelu Activation function.

The decoder has the same build, but instead uses transpose layers, which are deconvolution layers (where each layer has 16 3x3 kernels). These are transformations that go in the opposite direction of a normal convolution and up sample the input. We also use LeakyRelu just like the encoder. This will up sample the input back to 128x128 from 16x16. The final layer of the autoencoder uses the sigmoid activation function, so that the outputs are between [0,1], allowing us to interpret the output as a pixel intensity for the output grey-scale image. The loss function we chose was the binary cross entropy loss, since our pixel values are normalised to take values in [0,1] and our decoder model is designed to generate samples that take values in [0,1].

2.1.2 LeakyRelu vs Relu Activation Function

The LeakyRelu activation function is an activation function which works the same as ReLu when applied to positive inputs, but when the input is negative, it returns a small negative value proportional to its input. This avoids the dying Relu problem, which occurs when the ReLu activation function outputs a zero gradient for any input. This is caused by the weights updating in such a way that the gradient becomes 0, and so the gradient fails to flow back during back propagation as the output is always 0 [2]. It also generalizes much better for deep neural networks, which is another reason to use it [3].

2.1.3 Training the Model

We chose to split the training data into 20% validation, 20% testing, 60% training data, since these images contain labels which could be used for evaluation. We tested this model on both color and grey-scale images, and decided on grey-scale because they are smaller in size and save training and testing time. After training, we extract the encoder part of the network, and use this to predict the feature embeddings of images. We chose a batch size of 32 since it had the best tradeoff between speed and accuracy.

2.1.4 Testing

For our testing, we used the encoder to predict the feature embeddings of a left image and compared it to the feature embeddings of 20 random images (also predicted by the encoder), 1 of which is the correct label in our dataset. We did this for all 400 images in the testing set. We then compute the cosine similarity between this left embedding with all 20 right embeddings and output the 2 most similar images, along with the left image and the correct label of the left image. We chose cosine similarity because it ignores scale differences in magnitude, which are already removed from consideration having normalized the feature vectors.

2.2 Siamese Network

We explored Siamese networks, typically used in tasks like signature verification and facial recognition to compare similarities between two input images. These networks employ two identical subnetworks with shared weights to measure similarity or dissimilarity between image embeddings [9]. We focused on adjusting the models, incorporating transfer learning and different loss functions.

For all our Siamese models, we used a 70% training, 15% validation, and 15% testing split. We also used on the fly generators in our training to feed in our training data. This was done so we did not have to load all pairs/triplets into memory at once, and was much more computationally efficient. We generated dissimilar pairs by choosing a random image from the "right" folder and pairing it with an image from the "left" folder. For every model, we used a batch size of 32 since we found that increasing it further led to slower training but didn't provide a significant difference in validation loss. We also used early stopping in each model, stopping the training if the validation loss did not improve for 2 epochs. This was done to try to prevent the model overfitting to the training set.

2.2.1 CNN Architecture

Our initial model featured an input layer, a Conv2D layer with 64 filters (5x5 kernel, ReLU activation) for low-level feature extraction, followed by max-pooling to help focus on prominent features. Then, a second Conv2D layer with 128 filters (3x3 kernel, ReLU activation) for higher-level features, followed by another max-pooling layer. We flattened the 2D feature maps for the dense layer, converting features to a lower-dimensional representation, and applied L2 normalization for increased scale invariance. This setup aimed to create embeddings that represent image content so that similar images yield similar embeddings, while dissimilar images yield distinct embeddings when it is fed pairs of similar and dissimilar images.

2.2.2 Transfer Learning

We made another experimental change by employing transfer learning to address the challenge of limited data. With only 2000 images in our training set, neural networks typically require a substantial amount of data. Transfer learning offers an efficient solution, as it enables us to accelerate the training process by leveraging a pre-trained model's valuable knowledge [10]. In our implementation, we utilized a ResNet50 base model with ImageNet weights. We froze the weights up to the conv5 layers, leaving ResNet50's lower weights intact. This approach allowed us to capitalize on low-level features like edges, textures, and simple shapes captured by the lower layers [7]. For the higher layers, which capture more task-specific information, we fine-tuned them to adapt to our specific task. This strategy enabled us to attain effective image representations, and reduced training time by approximately 50%.

2.2.3 Loss Function

The choice of a good loss function is crucial for training Siamese networks. In our model, we need a loss function that can encourage the model to embed images in a way that similar images are close in the feature space, while dissimilar images are further apart. For our model, we tried two commonly used loss functions; triplet loss and contrastive loss.

Contrastive loss works by penalising the model when pairs of similar examples are far apart, and pairs of dissimilar examples are too close [11]. For our training batches, we feed the model pairs of images along with a label indicating whether they are similar or dissimilar.

$$\frac{1}{2} \times (1-y) \times D^2 + \frac{1}{2} \times y \times \max(0, margin - D)^2$$

Fig. 1. Contrastive loss equation where 'y' is the label (1 for similar, 0 for dissimilar), 'D' is the Euclidean distance between embeddings of the two images, and 'margin' which is a hyperparameter that controls how far apart the embeddings of dissimilar pairs should be.

Triplet loss takes in a training triplet (left image, similar, dissimilar) instead of pairs [11]. The loss measures the distance from the left image to the similar image and the distance from the anchor to the dissimilar image.

$$\max(0, D(a,p) - D(a,n) + margin)$$

Fig. 2. Triplet loss equation where 'D(a, p)' is the distance between the left image and the similar image, 'D(a, n)' is the distance between the left image and the dissimilar image, and 'margin' is the minimum difference required between the positive and negative distances.

For both loss functions, we used a margin of 0.5 as this provided the best results after experimentation.

2.2.4 Other Model Adjustments

Another approach we tried was hard negative mining, which involves choosing the most difficult images (where they appear more similar to the left image but arent actually the similar image) to feed in as dissimilar pairs. This should improve performance since it would improve model discrimination, allowing it to focus on challenging or misclassified negative examples. However, we were unable to evaluate its performance as it was far too computationally expensive to keep predicting over the 'right' folder to find the dissimilar image that was most similar to each 'left' image for every image pair.

III. Results

3.1 Autoencoder Results

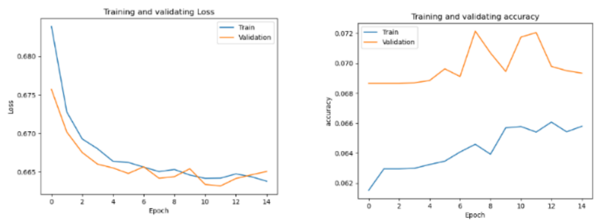

Fig. 3. Training and validation loss/accuracy over epochs

In figure 3, we see that the training and validation loss stay very close to each other as the model converges, while both training and validation accuracy see a slight increase each epoch.

Top x Accuracy	Accuracy (%)
1	17.75 (71 out of 400 images)
2	30 (120 out of 400 images)
3	35.25 (141 out of 400 images)
4	42.5 (170 out of 400 images)

Fig. 4. Top x Accuracy vs Accuracy

We also checked how close the models' outputs were to the correct output. In figure 4 we can see that many images are quite close to being labelled correctly, as 40% of the time, the correct label is within the top 4 similar images. For Kaggle, we got an accuracy of 29%.

3.2 *Siamese Network Results*

Model	*Test Accuracy (%)*
Self-Trained Feature Extractor	18.92
Transfer Learnt Features	21.46
Triplet Loss	20.97
Data Augmentation	20.83

Fig. 5. Test accuracy for each model

For each Siamese model, we used a similar task to the Kaggle competition to evaluate performance (selecting two most similar images from 20 images). We tested on each model to check if the performance improved for each change, and if it did improve, we included the changes for the other models. Here, we see that using transfer learning from the ResNet50 ImageNet model did improve the accuracy however, doing the same task with a triplet training and triplet loss function, and adding data augmentation did not cause any significant improvements in our models. For evaluating loss and for critical analysis, we only look at the transfer learnt model as it had the best results without any significant changes from the initial Siamese model.

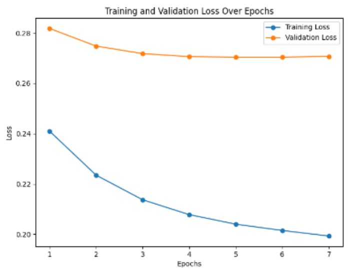

Fig. 6. Training and validation loss per epoch for transfer learnt Siamese network

For the transfer learning based Siamese model, we see that the training loss decreases per epoch, and that the validation loss also decreases, but plateaus at around epoch 5. There is also quite a significant gap between the training data loss and the validation data loss. This will be discussed in the critical analysis.

3.3 *Overall Model Results*

Overall, using our own test set and using the Kaggle test set, we see that the autoencoder approach presented significantly better results over all the Siamese models with a Kaggle accuracy of 29.2%, while the best of the Siamese models (the transfer learning model with contrastive loss and no data augmentation) only achieved a Kaggle accuracy of 21.8%.

IV. Critical Analysis

4.1 *Autoencoder*

When analysing our autoencoder's training epochs, we found that the validation and training loss was not improving much on each training run, only having a difference of 0.2 between the 1st and 15th epoch. This was concerning, as it was possible that the autoencoder was not learning the feature embeddings well enough to backpropagate useful gradients to the network.

4.1.1 *Training and Validation Loss/Accuracy*

From the training and validation loss graph, we can see that there is little evidence of overfitting as the validation and training loss graphs are similar to each other. However, we see that after only 4 epochs, the graph starts to plateau which suggests our model wasn't learning anymore. To try and fix this, we tried multiple things such as changing the activation to LeakyRelu, using less convolutional layers, and trying to change the batch sizes.

4.1.2 *Overall Performance*

We tried different top accuracy evaluations to see how far off our prediction is from the true label. After checking some of the accuracies in table 3, we can see that many images are quite close to being labelled correctly, as 40% of the time, the correct label is within the top 4 similar images. This shows that our model is learning some sort of useful feature embedding with every image as it is able to see some similarity between it and the target image. The top 1 and top 2 accuracy prove that the model needs to be fine-tuned in some ways to get a better accuracy.

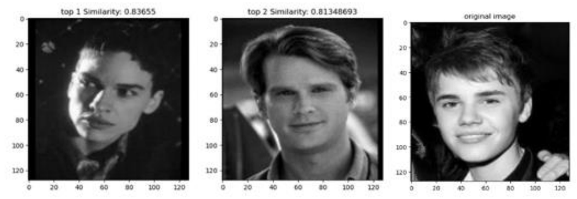

Fig. 7. Example smiliarity task where the network succeeds

When looking at individual images, we can further inspect what the network has learnt. From figure 7, we can see an example of a correctly guessed label, where the top 1 image is the correct label for the original image. We can see that the similarity score is fairly high between the target image and the correct label, meaning that the network is learning something valuable. The top 2 image also shows the features that our autoencoder is learning, as the person has the same pose and smile as the original image, as well as most of the same colour tones, which makes them similar.

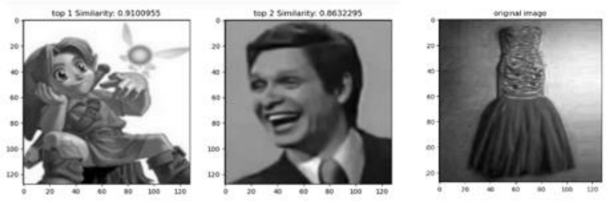

Fig. 8. Example smiliarity task where the network fails

Figure 8 shows some images which are consistently classified incorrectly, which mostly occur when an object is involved. We can see that in this case, there are no similarities between the target image and the top 1 or top 2 similarity images. The network seems to be learning some incorrect features, of something that has nothing to do with the images themselves. From these observations that we see through manually inspecting some test data, we can see that our network learns facial features and poses really well, but struggles to adapt to images which are only similar in shape.

4.2 *Siamese*

4.2.1 Transfer Learning

Analysis on our five different Siamese model implementations showed that using transfer learning did improve the accuracy of our image similarity task. This is because the feature representations from ImageNet are derived from a much larger dataset, so these representations for things such as edges or textures are much richer than the ones we extracted from our smaller dataset, and so we only need to train the Siamese network to fine tune the last layer to distinguish between similar and dissimilar images [7].

4.2.2 Loss Function

In our test on triplet loss vs contrastive loss, there was no significant change in accuracy. This is likely because these loss functions both aim to achieve the same goal, that is, making the embeddings for similar images closer in vector space, and making the embeddings for dissimilar images further apart in vector space, with the main difference being the data input that the models take in. Another reason the loss function may not have had any significant impact was that other issues such as overfitting were making the differences between loss functions less pronounced.

4.2.3 Data Augmentation

To attempt to improve our accuracy, we also tried data augmentation but it did not significantly affect our model. We attributed this to the fact that many images in the training data likely already had lots of variation, for example, we did horizontal flip as an augmentation, but many of the 'right' images from the data set are already horizontally flipped when compared to 'left' images. An example of this can be seen from one of the outputs on the test dataset, where the right image is very similar to a horizontally flipped version of the left image.

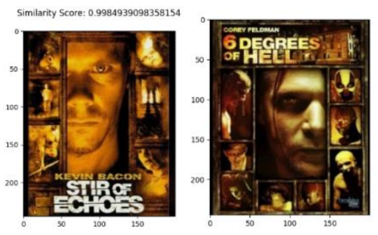

Fig. 9. Example smiliarity task where the siamese network succeeds

Our Siamese model demonstrates learning as both training and validation losses steadily decrease during training. It excels when processing images with similar structures and colours, as evident in most test examples with clear similarities. The model performs well when dealing with images containing distinct features at the same scale, as seen in the example with well-defined eyebrows (figure 9). However, it faces challenges with images that differ significantly in colour (e.g., when a "left" image has a noisy background while the "right" image has a plain white background). Additionally, it struggles when matching features of different scales.

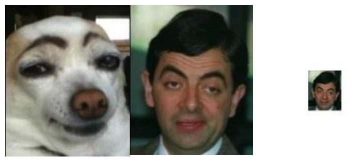

Fig. 10. Two images correctly classified, but the third smaller image is considered dissimilar by the model

We noticed that for an image in our test split of the training set, it constantly gave a low score to the matching image even though it was essentially the same image but scaled down. We suspected this may be to do with our network not being fully scale invariant, and we found this to be the case when we compared two images where one was an image of Mr Bean, and the other was the same image but scaled down onto a white background. The model did not perform well on this image, and classified a completely different image to be more similar. This suggests that variance in scale of the matching images was a problem for our Siamese model, or that our model was focusing too much on the background of the images and it wasn't finding good feature representations.

4.3 *Model Comparison*

Overall, we found that the autoencoder approach gave significantly better results. We see from the training and validation loss graphs, the autoencoder fit quite well to the validation set, while the Siamese network was greatly overfit to the training set. This could be due to several factors such as our choice of hyperparameters, or that the Siamese network was too complex. In terms of Kaggle performance, the autoencoder outperformed the Siamese network by about 8% accuracy. The autoencoder may have performed better as it essentially learnt to represent the images in a lower-dimensional feature space, capturing the most salient features of the input, while our Siamese model did not inherently use any dimensionality reduction and did not generalize well to testing data.

4.4 *Future Improvements*

Given our low accuracy, further tests with different hyperparameters and models hold promise for improved results. To address overfitting in the Siamese network, we can explore merging both approaches into a single model. Using an autoencoder to learn low-dimensional representations of input images, followed by inputting these encoded images into the Siamese network, could mitigate noise and enhance learning from essential features. Additionally, we might consider revisiting hard negative mining, selecting challenging examples from a smaller dataset subset to reduce computational demands. Exploring transfer learning with the autoencoder could also enhance feature representations, similar to its impact on our Siamese network.

V. Conclusions

Overall, we found that the autoencoder had much higher success than the Siamese network for our image similarity task with a Kaggle accuracy of 29.2%. It had little issues with overfitting, and changing the batch size to a smaller number helped our model learn more efficiently. We also found that our model is good at detecting facial features, while when objects are involved, there is less success.

For the Siamese network, we found that using transfer learning improved test accuracy as well as training time, while using triplet loss rather than contrastive loss, using data augmentation, and adding more dense layers had very little effect on the test accuracy, likely due to the strong overfitting. We also found that the Siamese network struggled with different scaled images but could generally capture strong similarities when the structure of the main features was very prominent, and when the backgrounds were not too varied.

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, “Title of paper if known,” unpublished.

[5] R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989. [1]K. Team, “Keras documentation: Image similarity estimation using a Siamese Network with a triplet loss,” Keras.io, 2021. https://keras.io/examples/vision/siamese_network/#:~:text=A%20Siamese%20Network%20is%20a (accessed Oct. 20, 2023).

[8] R. Gandhi, “Siamese Network & Triplet Loss,” Medium, Apr. 17, 2020. https://towardsdatascience.com/siamese-network-triplet-loss-b4ca82c1aec8 (accessed Oct. 20, 2023).

[9] S. B. J, “A friendly Introduction to Siamese Networks,” Medium, Jan. 29, 2021. https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942

[10] R. Andreoni, “Image Recognition Algorithm using Transfer Learning,” Medium, Sep. 03, 2023. https://towardsdatascience.com/image-recognition-algorithm-using-transfer-learning-cae1deb2818f (accessed Oct. 20, 2023).

[11] jdhao, “Some Loss Functions and Their Intuitive Explanations,” jdhao.github.io, Mar. 13, 2017. https://jdhao.github.io/2017/03/13/some_loss_and_explanations/#:~:text=Contrastive%20Loss&text=The%20loss%20function%20for%20a%20single%20pair%20is%3A (accessed Oct. 20, 2023).