COMP90086-GROUP PROJECT

Congcong Zhao 1107156 and **Fanhao Nie 1370013**
congcong@student.unimelb.edu.au
fanhaon@student.unimelb.edu.au

Abstract

In this project, we developed our custom neural network framework and introducing various neural network models to address the "Totally-Looks-Like" challenge. Moreover, we incorporated pre-trained models to extract image features and applied feature fusion techniques to tackle this specific problem.

1 Introduction

Modern computer vision algorithms differ from human capabilities in tasks like recognizing similar images. We chose a neural network framework and combine it with various models to handle the challenge. We also introduced feature fusion techniques in our experiments.

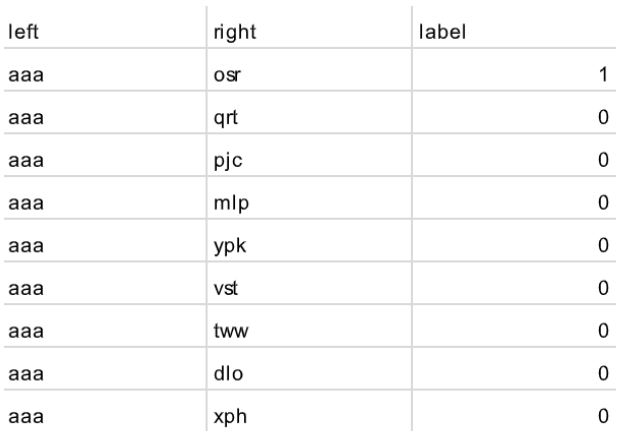
2 Method

To address the challenge of working with two similar images, our method that included data preprocessing, training method development, and dataset organization.

2.1 Data processing

In traditional supervised learning, labels are assigned to individual inputs. However, classifying each image across the entire dataset is overly complex. To address this, we've developed a method using one left image as a base and comparing it with twenty right images. Among these twenty, one is a true match, while the remaining nineteen are deliberately dissimilar, which facilitates the training process.

Each image within the data groups is labeled either 1 for similarity or 0 for dissimilarity. This process generates a *'random_train_data.csv'* file. This file is employed by the Custom DataGenerator class with a batch size of 32 and input image dimensions fixed at 200x245. Data is divided into an 80% training set and a 20% validation set. The structure of the CSV file is illustrated in Figure 1, showing the replication of one left image twenty times to match with corresponding right images.



left	right	label
aaa	osr	1
aaa	qrt	0
aaa	pjc	0
aaa	mlp	0
aaa	ypk	0
aaa	vst	0
aaa	tww	0
aaa	dlo	0
aaa	xph	0

Figure 1: The data structure in the "random_train_data.csv" file

2.2 Design of the model

Addressing the "Totally-like" challenge starts with a comprehensive dataset examination, which is crucial for creating diverse inputs that cater to the problem's unique aspects. The original training dataset is categorized into "left" and "right" folders, and notably, the only additional information available is from a train.csv file that provides insight into which right image bears the highest similarity to the corresponding left image.

2.3 Siamese Neural Network

The input of two images is precisely suitable for constructing a model based on Siamese neural networks for this challenge. According to the article Koch et al. (2015), the Siamese network comprises two twin networks that handle distinct inputs but share a common layer at the top. Then proper function is applied to calculate a metric using the highest-level feature representations on each side (illustrated in Figure 2). This approach ensures prediction consistency by tying weights, preventing similar images from being mapped to widely different locations in feature space. Moreover, the network's symmetry means that when two distinct images are input to the twin networks, the top layer computes the same metric.

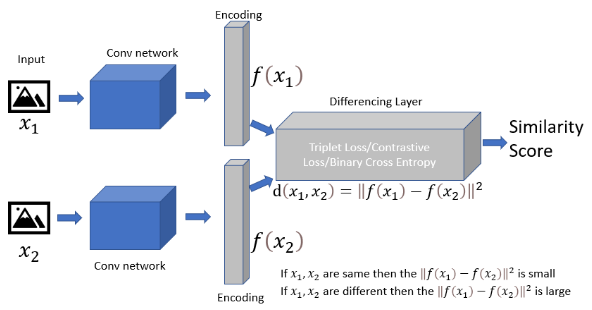

Encoding
Input
Conv network
x_1
$f(x_1)$
Differencing Layer
Triplet Loss/Contrastive Loss/Binary Cross Entropy
Similarity Score
$d(x_1, x_2) = \|f(x_1) - f(x_2)\|^2$
x_2
$f(x_2)$
Conv network
Encoding
If x_1, x_2 are same then the $\|f(x_1) - f(x_2)\|^2$ is small
If x_1, x_2 are different then the $\|f(x_1) - f(x_2)\|^2$ is large

Figure 2: Siamese Neural Network, input with two layers and share layers with same weight

3 Experiments

This section involves in the specific steps on how we train our models and try to get better results, including the model input, output, choice of loss function, adjustments of the model and results.

3.1 Model training

3.1.1 Siamese Neural Network based on Resnet18

- Model input: Based on the data structure defined in the *'random_train_data.csv'* file and the train set and validation set generated by the CustomDataGenerator data generator, the input receive a list which includes the left and right image vector.
- Model output: The output is the similarity between two image vector calculated from the sigmoid activation function
- Loss function: binary cross entropy is applied because the label defined only has 1 and 0
- Model implementation: Use TensorFlow library to construct resnet18 structure until the fully connected layer. Compile the model based on defined input, output and loss function with 10 epochs. Then pass sigmoid activation function using the difference between two image vectors. The output represents the similarity of the two images. The higher the score, the higher similarity it has between two images.

3.1.2 Siamese Neural Network based on VGG16

- Model input: The input receive a same format list as resnet18 model receives.' Train and validation set also generated by CustomDataGenerator.
- Model output: First try was based on relu activation and calculated the cosine similarity then. Subsequent attempt used the sigmoid activation.
- Loss function: Leave it unchanged, use binary cross entropy.
- Model implementation: Use the TensorFlow library to build the VGG16 structure. Defined a model containing 10 epochs. Use relu activation instead thereby outputting the similarity between the two images.

3.2 Evaluation

To assess the model's performance, our primary benchmark is the accuracy of the final results submitted to Kaggle. This metric reflects how effectively the model predicts image similarity.

3.2.1 Resnet18

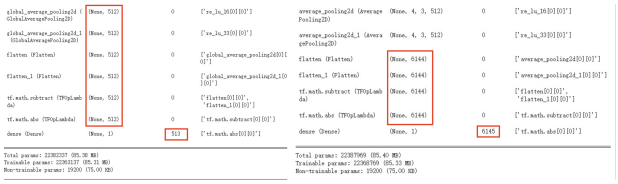
Create a Siamese Neural Network based on resnet18 and then flatten the features subtracting them and output the result by applying sigmoid activation function, so that it maps value 0 to 1. The higher the score, the higher similarity it has between two images.

The initial result is pretty low with only 0.091. According to a paper Putra and Setumin (2021), it points out that it actually uses the absolute difference to learn the similarity. If the difference between two vectors does not take an absolute value in a Siamese neural network, the result may be affected by the direction information, not just the size of the difference. The result then improved to 0.198 after using absolute difference.

In the original ResNet18 model, the last second layer employs global average pooling, which works well for classification. However, we opted to use an average pool instead to retain more local feature details in the image. This change led to an increase in results to 0.235, with the output parameters rising to 6144, approximately 12 times more than with global average pooling (see Figure 3).

3.2.2 Resnet18 error analysis

We opted for the best-performing model among the three trained models and assessed its



```
global_average_pooling2d (        (None, 512)        0        ['re_lu_16[0][0]']
GlobalAveragePooling2D)

global_average_pooling2d_1        (None, 512)        0        ['re_lu_33[0][0]']
 (GlobalAveragePooling2D)

flatten (Flatten)                 (None, 512)        0        ['global_average_pooling2d[0][
                                                               0]']

flatten_1 (Flatten)               (None, 512)        0        ['global_average_pooling2d_1[0
                                                               ][0]']

tf.math.subtract (TFOpLamb        (None, 512)        0        ['flatten[0][0]',
da)                                                             'flatten_1[0][0]']

tf.math.abs (TFOpLambda)          (None, 512)        0        ['tf.math.subtract[0][0]']

dense (Dense)                     (None, 1)          513      ['tf.math.abs[0][0]']

==================================================================================================
Total params: 22382337 (85.38 MB)
Trainable params: 22363137 (85.31 MB)
Non-trainable params: 19200 (75.00 KB)
```

```
average_pooling2d (Average        (None, 4, 3, 512)        0        ['re_lu_16[0][0]']
Pooling2D)

average_pooling2d_1 (Avera        (None, 4, 3, 512)        0        ['re_lu_33[0][0]']
gePooling2D)

flatten (Flatten)                 (None, 6144)             0        ['average_pooling2d[0][0]']

flatten_1 (Flatten)               (None, 6144)             0        ['average_pooling2d_1[0][0]']

tf.math.subtract (TFOpLamb        (None, 6144)             0        ['flatten[0][0]',
da)                                                                   'flatten_1[0][0]']

tf.math.abs (TFOpLambda)          (None, 6144)             0        ['tf.math.subtract[0][0]']

dense (Dense)                     (None, 1)                6145     ['tf.math.abs[0][0]']

==================================================================================================
Total params: 22387969 (85.40 MB)
Trainable params: 22368769 (85.33 MB)
Non-trainable params: 19200 (75.00 KB)
```

Figure 3: Using global average pooling with different output parameter 513

accuracy and loss to monitor the training progress. We observed that the training accuracy plateaued after approximately 10 epochs. Similarly, the validation accuracy displayed a similar trend, with some fluctuations between the third and fourth epochs.

However, when analyzing the training and validation loss, we noticed an increase in the validation loss after the eighth epoch, reaching a loss value of 0.26.

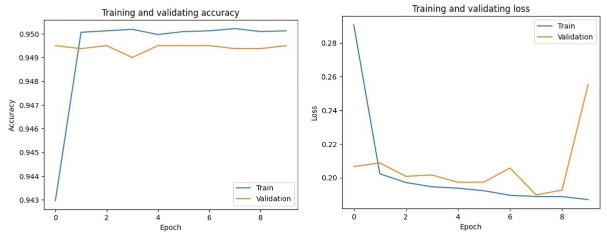


Figure 4: This is the Resnet18 model accuracy and loss performance

3.2.3 VGG16

Constructed a Siamese neural network based on the VGG16 model. Flattened the features and employed the relu activation function to produce the final output.

The initial training results were unsatisfactory, yielding only a score of 0.111. After analyzing the model and conducting investigations, the learning rate was adjusted following the guidance from a referenced paperQassim et al. (2018), increasing it from the default 0.001 to 0.005. However, the second result remained around 0.11. Considering the size of the dataset, we thought that the main reason is due to the quality of the data.

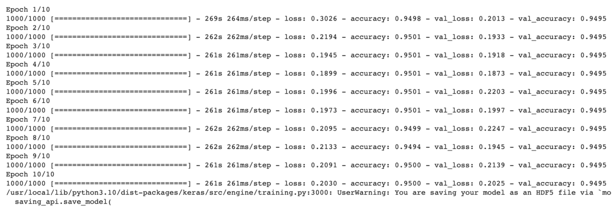
Subsequently, inspired by experiments with the ResNet18 model, absolute differences were utilized to learn similarities. This adjustment still failed to improve performance very well.

3.2.4 VGG16 error analysis

Due to time constraints, the current adjustments to the VGG16 model we built have not improved very well, and the final performance only improved to 0.198 when we use sigmoid activation. We analyzed the model and monitored the accuracy and loss during training to help us conduct further experiments in the future.

It can be observed that under the premise of limiting 10 epochs, the accuracy of the model tends to be stable after the first epoch, with only small fluctuations occurring during the 7th and 8th epochs. And the verification accuracy is very stable.

For the training loss, we can observe that it stabilizes at around 0.20 after declining in the first epoch. For the verification loss, it is stable and fluctuates slightly around 0.20 in the long term.



```
Epoch 1/10
1000/1000 [==============================] - 269s 264ms/step - loss: 0.3026 - accuracy: 0.9498 - val_loss: 0.2013 - val_accuracy: 0.9495
Epoch 2/10
1000/1000 [==============================] - 262s 262ms/step - loss: 0.2194 - accuracy: 0.9501 - val_loss: 0.1933 - val_accuracy: 0.9495
Epoch 3/10
1000/1000 [==============================] - 261s 261ms/step - loss: 0.1945 - accuracy: 0.9501 - val_loss: 0.1918 - val_accuracy: 0.9495
Epoch 4/10
1000/1000 [==============================] - 261s 261ms/step - loss: 0.1899 - accuracy: 0.9501 - val_loss: 0.1873 - val_accuracy: 0.9495
Epoch 5/10
1000/1000 [==============================] - 261s 261ms/step - loss: 0.1996 - accuracy: 0.9501 - val_loss: 0.2203 - val_accuracy: 0.9495
Epoch 6/10
1000/1000 [==============================] - 261s 261ms/step - loss: 0.1973 - accuracy: 0.9501 - val_loss: 0.1997 - val_accuracy: 0.9495
Epoch 7/10
1000/1000 [==============================] - 262s 262ms/step - loss: 0.2095 - accuracy: 0.9499 - val_loss: 0.2247 - val_accuracy: 0.9495
Epoch 8/10
1000/1000 [==============================] - 262s 262ms/step - loss: 0.2133 - accuracy: 0.9494 - val_loss: 0.1945 - val_accuracy: 0.9495
Epoch 9/10
1000/1000 [==============================] - 261s 261ms/step - loss: 0.2091 - accuracy: 0.9500 - val_loss: 0.2139 - val_accuracy: 0.9495
Epoch 10/10
1000/1000 [==============================] - 261s 261ms/step - loss: 0.2030 - accuracy: 0.9500 - val_loss: 0.2025 - val_accuracy: 0.9495
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file via `mo
  saving_api.save_model(
```

Figure 5: This is the VGG16 model accuracy and loss performance

3.3 Pre-trained model

According to this article Rosenfeld et al. (2018), it points out a workflow for solving this TLL challenge. The basic idea here is to use pre-existing or pre-trained model to extract the general features of the image and there is an adaptation modules where we can use several ways to.

3.3.1 Generic feature extraction

I performed a comparison of cosine similarity based on features extracted from various layers to evaluate their impact on similarity. It was noted that while earlier layers performed well for some pairs, their performance deteriorated for different image pairs. Initially, the test set yielded a score of 0.34 when utilizing features up to the last fifth layer.

As a result, I decided to exclude the fully connected layer and instead employed features from the first layer to the last pooling layer, aligning with the ResNet model's architecture. This adjustment led to an improved similarity score of 0.443. Subsequently, I repeated the same steps using ResNet 50 and achieved a similarity score of 0.48.

The original VGG16 pre-trained model initially performed well, achieving a result score of

0.504. In pursuit of better results, I fine-tuned the model's hyperparameters and attempted to enhance performance by using the Adam optimizer. Simultaneously, I made adjustments to the model's last fully connected layer, changing the initial 4096 output units to match the number of output units required for num_classes. However, these modifications did not lead to an improvement in the final result, which remained at 0.504.

3.3.2 Feature fusion

Upon dataset analysis, we noticed that most images predominantly featured human faces. Therefore, we incorporated and extracted facial features in addition to general image features. This was achieved using the dlib library for facial feature extraction. We defined a facial scanning area covering the entire image and assigned a weight of 66% to the facial features, while generic features received the remaining weight.

This approach led to an improved similarity score of 0.447. Additionally, the VGG16 model's score saw a slight increase, reaching 0.505.

4 Conclusion

In summary, we constructed a Siamese neural network framework based on ResNet18 and VGG16 to address the challenge of similar image recognition. We fine-tune the built model such as adjusting parameters, adding more layers, etc. to obtain better results. At the same time, we also introduce feature fusion and concatenate to extract and compare more specific features such as faces and textures to increase the prediction accuracy of the model. In order to achieve better performance of the model, we also preprocessed and classified the images in the data set, and trained the model using different methods.

The final best-performing model was the pretrained VGG16 model using the relu activation function and facial feature extraction, which reached 0.505. The best-performing manually built model is the ResNet18 model using average pooling and absolute difference, reaching 0.235.

5 Future improvement

Considering our current implementation, we acknowledge the fact that we still need improvements to tackl the TLL challenge more effectively. We have identified potential future enhancements and established certain assumptions to strengthen our strategy for addressing the TLL challenge.

5.1 Train model - Model design

Categorizing all images in the entire dataset is complex and time-consuming. If we can group images into subcategories, we can explore alternative training methods. This might include modifying the loss function, altering input and output formats, using image vectors for input, obtaining image labels as output, and introducing extra layers to enhance classification training and extract image features for similarity assessment.

5.2 Pre-trained model - Fine tuning

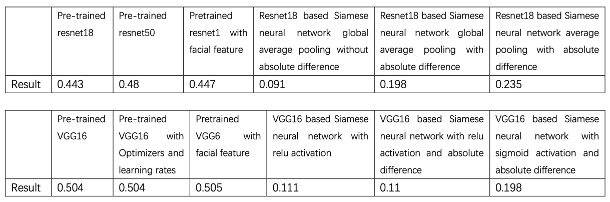
To improve the model's performance, we can consider adding more layers or modifying existing ones to capture finer details in the data. This can be achieved by fine-tuning specific layers while keeping others frozen. We can then retrain the updated layers using the current dataset.

Furthermore, we can explore different similarity measures, such as Euclidean distance, to compare the feature vectors extracted from the images. This may provide an alternative perspective on similarity and help enhance the model's ability to distinguish between images.

5.3 Feature fusion

We can investigate advanced feature extraction methods, such as isolating key objects within the images, and then combining these features with other elements like facial attributes, textures, or any relevant characteristics.

6 Result



	Pre-trained resnet18	Pre-trained resnet50	Pretrained resnet1 with facial feature	Resnet18 based Siamese neural network global average pooling without absolute difference	Resnet18 based Siamese neural network global average pooling with absolute difference	Resnet18 based Siamese neural network average pooling with absolute difference
Result	0.443	0.48	0.447	0.091	0.198	0.235

	Pre-trained VGG16	Pre-trained VGG16 with Optimizers and learning rates	Pretrained VGG6 with facial feature	VGG16 based Siamese neural network with relu activation	VGG16 based Siamese neural network with relu activation and absolute difference	VGG16 based Siamese neural network with sigmoid activation and absolute difference
Result	0.504	0.504	0.505	0.111	0.11	0.198

Figure 6: These two tables show the final result we got

References

Koch, G., Zemel, R., Salakhutdinov, R., et al. (2015). Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille.

Putra, A. A. R. and Setumin, S. (2021). The performance of siamese neural network for face recognition using different activation functions. In *2021 International Conference of Technology, Science and Administration (ICTSA)*, pages 1–5. IEEE.

Qassim, H., Verma, A., and Feinzimer, D. (2018). Compressed residual-vgg16 cnn model for big data places image recognition. In *2018 IEEE 8th annual computing and communication workshop and conference (CCWC)*, pages 169–175. IEEE.

Rosenfeld, A., Solbach, M. D., and Tsotsos, J. K. (2018). Totally looks like-how humans compare, compared to machines. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1961–1964.