IDs:

ID #1: 300822954

ID #2: 307963538

Deep Learning

Assignment 2 – Siamese Neural Networks and one-shot learning

As instructed, our implementation is based on the paper “Siamese Neural Networks for One-shot Image Recognition”.

In this assignment, we extend and explored various architectures, methods (such as transfer learning), etc. to train a Siamese NN model based on [Labeled Faces in the Wild](http://vis-www.cs.umass.edu/lfw/index.html) dataset that able to perform face image verification task i.e., given two faces’ images decides if those two belong to the same person.

The general idea is that the network learns to naturally rank similarity between faces images inputs, aka, similarity score, thus, later on, the network can generalize the prediction and employ it in a zero/one-shot learning fashion.

The hypothesis for Siamese NN is that if two input faces images belong to the same person, then their feature vectors must also be similar, while if the two input images belong to different persons, then their feature vectors will also be different. Hence, the element-wise absolute/euclidean-distance difference between the two feature vectors must be very similar/different and the similarity score generated by the output sigmoid layer must also be similar/different for the above cases in which this type of learning method is different than learning to classify an image directly to any of the output possible persons.

Moreover, multiple experiments have been conducted such as examine various network architectures, different learning rates, different dense layer sizes, augmented the dataset by applying transformations, applying regularization methods, etc. By doing so, we present below and explain the effect and sensitivity of the model’s performance caused by varying values of the hyper-parameters, different architectures, and methods and show the effect on the training process and performance.

Final model configuration and score:

TODO

**Experiments setup:**

Pre-processing and dataset pipeline steps:

Note:

1. Our logic for the pre-processing and dataset pipeline is located under preprocessing\_utils.py script
2. We decided to use a validation set by splitting the given 2200 training pairs with validation\_size=0.2 which represents the proportion of the dataset to include in the validation split. It is important to note that although the given training dataset is relatively small and our models overfitted, we decided to mitigate this issue by other methods such as data augmentation and regularization methods. Moreover, we tried to train a model with the whole given training dataset which still produces an overfitted model.

Steps:

1. LFWA dataset was downloaded, which contains directories with face images per person. Also, includes ‘pairsDevTrain.txt’ and ‘pairsDevTest.txt’ files that determine the training/test datasets. These files hold pairs of matching/non-matching images.
2. Parse each file and produce a relevant data structure that holds the matching and the non-matching records.
3. Split the given training pairs into train/validation datasets – the split was performed on the matching and non-matching pairs separately, to ensure each set contains samples of both types of pairs, and then the validation/training set of the non-matching pairs was united with the appropriate matching pairs set (validation-matching with validation-non-matching and similarily with the training sets of pairs).
4. Convert the current data structures that hold images path into tensors with the content of the images by decoding, apply image resize if configured, and normalization.
5. We decided to work with Tensorflow Dataset API, thus, converted the above pair's images data structure into training/validation/test into tf.data.Dataset and configured it by applying if necessary per dataset type: cache, shuffle, batch and prefetch.

The models were obtained by a training procedure which was bounded by 50 epochs and early stopping that stop training when the binary accuracy on the validation has stopped improving with patience=15 and min\_delta=0.03. Also ‘restore\_best\_weights’ flag was set to True.

To monitor performance during training, we used the binary accuracy metric, namely, verification on the validation set pairs generated as depicted above. We could have chosen to apply n-way one-shot learning evaluation, however, in the paper they stated that both strategies yield similar results, thus for ease of implementation, we’ve decided to configure our optimizer and early stopping based on validation error for the verification task.

However, we also provide details with respect to an n-way one-shot learning evaluation on the tuned model evaluated on the validation and test dataset.

For performing n-way one-shot learning evaluation we added logic that generates those tests for both validation and test dataset into the above dataset pipeline.

Our logic creates per pair in the dataset under the matching section n-way test, namely, the same image is compared to n different images out of which only one of them matches the original person. Specifically, for each matching pair, we first search for pairs associated with the relevant person in the non-matching section, and in case that no enough n-1 cases exist, we randomly choose non-matching person images from the non-matching section.

In our configuration, we set n=3 and we didn’t include it in our hyper-parameter search space. It is important to note that larger values of n will lead to relatively less correct predictions.

All training data was saved into Tensorboard as can be seen below in the empirical results section.

We wrap our experiments with Bayesian optimization to perform hyperparameter selection. We utilize the Keras-Tuner framework and configured it as follows:

Max\_trials=for some experiments it was set with 150, for others, 50 was configured

Objective=max ‘val\_binary\_accuracy’

Num\_initial\_points=3

With hyper-parameters as follows:

'learning\_rate': [5e-5, 1e-5, 5e-4, 1e-4, 1e-3, 5e-6]

'dense\_layer\_size': [1028, 4096, 512]

'enable\_batch\_normalization': [True, False]

'bias\_initializer': ["default", "zeros"]

'conv2D\_kernel\_initializer': ["default", "he\_normal"]

'dense\_kernel\_initializer': ["default", "he\_normal"]

'dropout\_rate': [0.0, 0.2, 0.5]

'distance\_metric': ["abs", "euclidean\_distance"]

'l2\_regularizer': [-1.0, 0.01, 0.05, 0.1, 0.5]

'optimizer': ["adam", "sgd", "RMSprop"]

Note: we fixed batch\_size=64 for all experiments. We set it to 64 and not to 128 since it worked well and provide good results and mitigate our issues with OOM during training due to the large image dimension and limited hardware.

As can be derived from the above search space, we enabled per experiment settings as demonstrated in the empirical result section below various inner model architectures and regularization methods such as if to apply batch normalization, dropout, l2 regularization, etc.

Also, regarding weight initialization, for “default” mode above, we followed the authors' suggestion in the paper, namely, for the convolutional layers the weights are initialized from a normal distribution with zero-mean and standard deviation of and for biases, the weights are initialized from a normal distribution with mean 0.5 and standard deviation .

and for the full- connected layers, the weights were drawn from a much wider normal distribution with zero-mean and standard deviation of and the biases were initialized in the same way as the convolutional layers.

We stored the results including the training log, model performance in Tensorboard hparams table, and csv file. Also, all trials and their final model are saved as well.

Remark: due to file sizes and submission limitations, we couldn’t submit all this information, thus, selected files were chosen.

Link to full logs and information can be found on the following link:

**TODO**

**Models’ Network Architecture:**

As specified in the assignment, each model was

**Siamese NN implementation details:**

We followed the

Hypermodel - TODO

**Data Augmentation**

Although in the paper the authors augmented the training set with small affine distortions we started our experiments without performing any kind of augmentation to the given training set, since those kinds of transformation to our opinion are less suited for our dataset.

However, our models suffer from overfitted, namely, training binary accuracy is 1 while validation binary accuracy is ~0.7.

Thus, we've decided to augment the training dataset with the following transformations:

1. Salt and pepper noise
2. Rotation by [45, -45] randomly
3. Center crop (0.5% center) and resize with padding
4. Flip left right
5. Salt and pepper noise + center crop

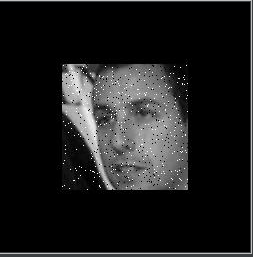
To combine data augmentation as part of our training procedure, we first perform offline one time per image in the training dataset all above transformations and saves the new images into relevant directories. Then, as part of our dataset pipeline, we added all the above augmented images to be part of the training dataset only, excluding validation/test datasets.

Examples:

Adam\_Sandler\_0002.jpg:

Original center crop flip left-right noise

Rotation-45 noise + center crop

Note: our logic for the data augmentation is located under the data\_augmentation.py script

TODOs list:

Architecture:

Training strategy: binary accuracy instead of one-shot during training??.

Learning rates: Also, an exponential learning rate decay mechanism has been applied, i.e., .

Optimizations:

Weight initialization

Distance

Data augmentation

Transfer learning

Regularization methods including batchnorm

Stopping criteria:

Reasoning behind the choices:

Convergence time, final loss and accuracy on test and holdout

Graphs

Performance

Example of accurate and misclassification and try to determine why

**Section 2C – EDA:**

We started our EDA by verifying that no subject )which we didn’t know whether it refers to pairs or individuals) is shared between the training and testing sets, as stated in the instructions. This was verified to be true for individuals and, as a an outcome, for pairs as well. From visually inspecting the image files, we also noticed that aside from centering and cropping the images to contain the individuals’ faces, some images were also cropped and rescaled, adding a completely black background in the edges of the frame.

To asses the scale of the provided image set, we checked how many image files each individual has in the image (‘Ifw2Data’) directory supplied and plotted the histogram of the individual counts of image files (Fig. 1). We discovered that the majority of individuals have only a single image file, while some individuals have hundreds of image files, with the maximum amount saved under ‘George\_W\_Bush’ name, with 530 images.

The individuals appearing in the image files themselves were also of interest: After browsing through hundreds of images (through the process of performing the assignment) we noticed that most, if not all, individuals were of a narrow demographic spectrum, no images of children and very old people seem to exist, most images appear to be of white middle aged males. The database webpage [itself](http://vis-www.cs.umass.edu/lfw/) seems to agree and even provides a friendly warning that this database does not fit well for the 1:N recognition task.

Chart, histogram

Description automatically generated

Figure :Distribution of Image files count - limit of y-axis is set to 10. Maximum count of image files detected is 530.

We then focused on the training and testing set only, disregarding any finding related to the image files themselves, as we are instructed to train and test our data using only the images addresed by the ‘pairDevTrain.txt’& ‘pairDevTest.txt’; We found that the individual appearing in the training set and in the test set are ‘Alec\_Baldwin’ ‘Tang\_Jiaxuan’ , accordingly, both with 6 appearances only (both under matching pairs and non matching pairs). We also plotted a histogram of all individuals’ amount of appearances in the training and testing sets (Fig. 2), and although most individuals appeared once, it seems there is some sort of a power law restricting the amount of duplicated (or more) inviduals’ appearances in both training and testing sets. In the training set, only 351 individuals appeared in both the matched pairs and non matched pairs, whilst 138 appeared in both matching and non matching pairs in the testing set.

A picture containing icon

Description automatically generated

Figure - Distributions of appearances in training and testing sets. The total number of unique individuals in the training set is 2132, the number of unique individuals in the matching pairs of the training set is 788, and in the non matching pairs 2132. The total number of unique individuals in the testing set is 963, the number of unique individuals in the matching pairs of the testing set is 353, and in the non matching pairs 748.

Checking for shared individuals across matching and non matching pairs is not sufficient, as an individual might be represented by a different image (notated with the image index) in the matching samples then in the non matching samples. Hence, we checked whether the set of images used for the matching samples is fully/disjoint/contiguous with the set of images used for the non-matching sample with all shared individuals. We found that all the images of individuals appearing in both matching and non matching pairs are the same in the matching and non matching samples.

EDA conclusions:

As most individuals appear in either the matching pairs or the non-matching pairs, the task in hand (one-shot learning to classify if two images are of the same individual) is more difficult than we initially thought. During most of the training, our model will learn from each individual only once, either from matching samples or non-matching samples, and will have very few opportunities to learn matching and non matching samples of a single person. This makes the task of training closer to zero-shot learning. This is emphasized by the fully disjoint train and test sets, were no individuals are contiguous about the two sets.   
All of the above and the disclaimers mentioned at the beginning of this section, suggest that in “real world” research, were the performance of the model actually matter, one should not use this dataset on its own as it might (among other issues) cause a bias towards specific demographics (e.g. it might always classify two images of children as a matching pair).

Stopping criterion:

**Empirical results:**

Image resize:

One of the hyper-parameters we tested for was whether to perform image resizing, where images are scaled to a 150x150 pixels matrix instead of the original 250x250 pixels matrix. The trails (in regards to this hyper parameter) are aimed to answer two concerns: (1) is the amount of information retained in a reduced size image sufficient for the task in hand, i.e. will the model be able to learn the important facial features that discriminate between individuals, or is the information encoding said facial features lost when images are scaled down in size. (2) Out of memory (OOM) issues; the domain of image processing is very resource demanding by its nature, so reducing the physical amount of memory each image occupies might accelerate training and testing processes.S

**Data augmentation**

Accuracy plots:

batch\_size=16 batch\_size=32 batch\_size=64

**The effect of the different batch sizes on training performance/convergence**

In this experiment, we look at the effect of using different batch size values on model performance/convergence.

**Results:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Batch\_size | Epoch | Iterations | Train accuracy | Test accuracy |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

As can be seen in the table above,

Note:

**The effect of the different learning rate on training performance/convergence**

**Remarks, Observation, and Comparison:**

1. From our experiments when

**Conclusion:**

The

**How To Run:**

1. open a terminal and cd to ‘HW1\_300822954\_307963538’ directory

2. conda create --name HW1\_300822954\_307963538 python=3.8

3.WINDOWS: activate HW1\_300822954\_307963538

LINUX, macOS: source activate HW1\_300822954\_307963538

4. pip install -r requirements.txt

For running our code, run

Default configuration: