## 1. Assignment goal

In this assignment, we implemented Siamese neural networks for one-shot image recognition task, based on paper [1]. We train the network over Labelled Faces in the Wild dataset [4], and let the model determine whether two given unseen images present the same person or not.

## 2. Environment setup

Deep learning modules: TensorFlow 2, Keras
Image processing: CV2, PIL
computing resources: Google Colab, 25 GB, GPU enabled
assignment code is available both on jupyter notebook and python formats.

## 3. The Model Architecture

**Overview**

The Siamese Network contains two identical convolutional networks – twins. Each twin receives as an input a single image which is a part of an example pair and produces as an output a features vector.

The Siamese Network performs an absolute distance calculation using a Lambda layer, over the two features vectors that were produced by each twin. The result of the calculation is received as an input to a final fully connected layer, that performs a sigmoid activation function and produce a prediction vector that indicates the two images similarity.

While building the architecture, Keras.IO example of MNIST siamese [2] helped us to understand better the flow and the process of siamese network implementation.

**Model structure**

During the network setup, we created the initial model by following the structure that described on [1]. After examining the model's performance, we made few changes and tuning on the initial structure and parameters. The model contains five convolutional layers, as summarized on the following table. More details are shown on figure 1.

| Layer | Shape dims | Nodes | Activation function |
|---|---|---|---|
| Input | 105,105,3 | | |
| L1 - conv | 10,10 | 64 | Relu |
| L2 - conv | 7,7 | 128 | Relu |
| L3 - conv | 4,4 | 128 | Relu |
| L4 - conv | 4,4 | 256 | Relu |
| L5 - Dense | | 4096 | Sigmoid |

As described on the paper [1], on each of the convolutional layer outputs we performed max pooling with a filter size of 1 and stride of 2.

The output values of the final convolutional layer are flattened into a vector that represents the features of the input image, and a Lambda layer computes the distance between those two features vectors. Finally, a sigmoid function is performed on the result and produces a prediction vector.

```
__________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_3 (InputLayer)         [(None, 105, 105, 3) 0
__________________________________________________________________________________________________
input_4 (InputLayer)         [(None, 105, 105, 3) 0
__________________________________________________________________________________________________
sequential_1 (Sequential)    (None, 4096)              38962752    input_3[0][0]
                                                                   input_4[0][0]
__________________________________________________________________________________________________
lambda_1 (Lambda)            (None, 4096)              0           sequential_1[1][0]
                                                                   sequential_1[2][0]
__________________________________________________________________________________________________
dense_3 (Dense)              (None, 1)                 4097        lambda_1[0][0]
==================================================================================================
Total params: 38,966,849
Trainable params: 38,965,697
Non-trainable params: 1,152
__________________________________________________________________________________________________
```

Figure 1:model summary

**Dropout** We decided to perform a dropout with a rate of 0.3 on the first layer output, and a dropout with a rate of 0.02 on the last layer output, in order to generalize the model and avoid overfitting. We notice to an unexpected result when performing dropout on the next convolutional layers output. As this article indicates and demonstrates [5], using dropout on convolutional networks may damage the learning process since it does not have the desired effect as on fully connected network.

**Model Width** We also examined different layers widths (number of nodes) to see whether a complex model with more parameters can perform better. However, it did not show improvement of the model performance and metrices.

**Batch normalization** We tried to improve model performance and accuracy by performing batch normalization after each layer.

## 4. Parameters Initialization

The main challenge we faced while training and validating the model was overfitting. Limited number of training examples on the training set (~2000), leads us to consider several ways in order to overcome this issue. While using dropout technique showed a little improvement, we needed further tuning in order to generalize the model.

**Weights and biases**

We initialized weights and biases as described on the paper, the values generated from a random normal distribution having mean of 0.0 (weights), 0.5 (bias) and std of 0.01 (0.2 for weights of final layer).

**Regularizations**

For weights regularization we used L2 regularization as described on the paper. We examined several values for L2 regularization parameter [0.001 - 0.1], and eventually set it to relatively small value of 0.001.

**Learning rate**

We used decay learning rate as described on the paper, reduced each epoch by 1%.

**Momentum**

We set the SGD momentum to a fixed value of 0.7. we noticed that large values tend to make large "jumps" during SGD, but when using Nesterov acceleration, we got better accuracy.

**Optimizers**

We tested several optimizers (Adam, Nadam, Rmsp, SGD). We noticed that GSD performed best for high learning rates, while the others require smaller learning rates to show improvement. We also noticed that batch normalization has some affect when using adaptive methods. According to [3], using adaptive methods may show in some cases worst generalization, comparing to SGD.

**Epochs** We set fixed value of 50.

**Batch size**

64,128. We tried several options for batch sizes and noticed that learning process is most stable and effective for those values. We kept fixed value of 64 during our experiments.

## 5. Image Preparation

We performed some common pre-proccessing techniques on input images.

**Reshape** We tried various input shapes before setting the final to 105X105, as suggested on the paper [1]. The initial shape 250X250 was too large for model processing and required massive computing resources. We also tried input shape 200X200, but it did not show improvement on accuracy, comparing to 105X105.

**Normalization** We performed a normalization to the range 0-1 by dividing the images matrix values in 255.

## 6. Loss function

We examined model's behaviour for two different loss functions that may fit for one-shot task.

**Constructive loss** We used constructive loss as presented here [7] and is also used on Keras example [2].

**Binary cross entropy** We used Keras binary cross entropy loss, which is very similar to the regularized cross-entropy that was presented in the paper [1]. Using the binary cross-entropy loss, we can measure how far the predicted value is from the true value.

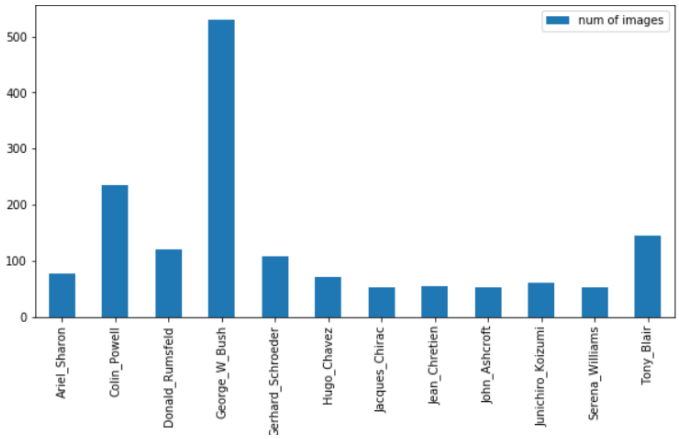
## 7. Train, Validation and Test data sets split considerations

During our initial experiments, we followed the original sets division as described on the LFW dataset paper [4]. We assumed that randomly splitting the given training set and use it for both training and validation purpose, might have some negative correlative effect, since we cannot control classes overlapping between the two groups. For training and validation, we used pairsDevTrain.txt (train) and pairsDevTest.txt (validation) as suggested on View 1. As described on the dataset paper, this view of data was created for development and parameters setting purpose. For model evaluation we used pairs.txt [6] as test examples as suggested on View 2. This view was created for performance reporting and evaluation.

However, while continuing development, we realized that specifically for this assignment it is not recommended to use external resource of test examples - since our model could not be evaluated compering to others, if we test different dataset. Eventually, we decided to keep the original assignment resources: we randomly split training data set (pairsDevTrain) to 80% train and 20% validation and evaluate our model's performance with pairsDevTest dataset.

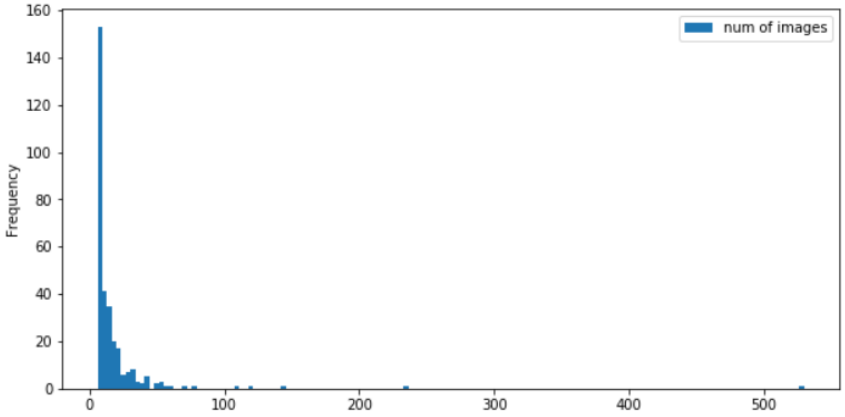
## 8. Dataset Analysis

In this assignment we used LFW dataset [5], a labelled dataset of people that was created for face recognition tasks. The dataset contains 13,234 images of 5,750 different people. 12 people has more than 50 images (graph 1). We can have a better insight by looking on following graphs.



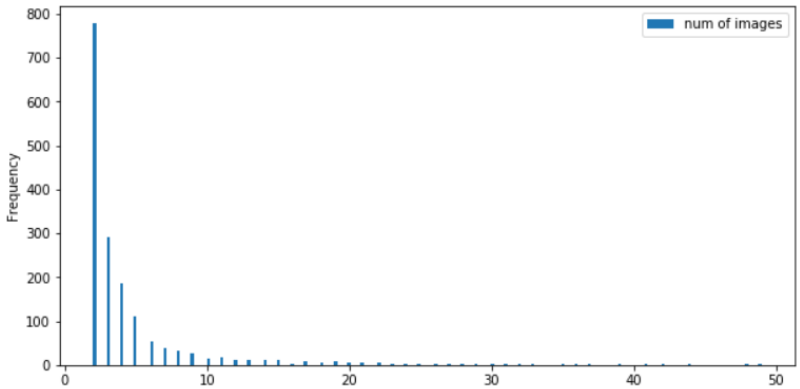

Graph 1: people who have more than 50 images

For people with more than 5 images, the following histogram shows us that the main majority has less than 50 images per people.




Graph 2: People with more than 5 images

Here we can get a closer look. For people with less than 50 images, the following histogram shows that most people have one (70%) or two (13.5%) images (4070 – 1 image, 779 – 2 images). This indicates that almost 85% of the classes (people) on our data set has very small representation.




Graph 3: people who have more than 50 images

**Train dataset**

Has total 2200 examples (pair of images). 1100 positive examples – two images belong to the same person, and 1100 negative examples, where two images belong to two different people. There are 788 different people whom images belong to positive examples, and 1695 different people whom images belong to negative example. We can see, who appears most of the times on our positive and negative train examples:

```
Angelina_Jolie 5
Catherine_Zeta-Jones 5
Jason_Kidd 5
Alec_Baldwin 4
Ana_Guevara 4
Ari_Fleischer 4
Bob_Graham 4
Cherie_Blair 4
Jennifer_Garner 4
John_Snow 4
Paul_Wolfowitz 4
Queen_Elizabeth_II 4
Ricardo_Lagos 4
Sachiko_Yamada 4
Tiger_Woods 4
Zhu_Rongji 4
Adam_Sandler 3
Amelia_Vega 3
Andy_Roddick 3
Barbara_Walters 3
Billy_Crystal 3
Bulent_Ecevit 3
Cesar_Gaviria 3
```

*Figure 3:Top appearance for positive examples*

```
Lin_Yung_Hsi 5
Ahmed_Ibrahim_Bilal 4
Hushiar_Zebari 4
Cristian_Barros 4
Wendy_Kennedy 4
Ray_Halbritter 4
Ricardo_Lopez_Murphy 3
Ray_Morrough 3
Elijan_Ingram 3
Jeff_Weaver 3
Li_Changchun 3
Azmi_Bishara 3
Bill_Rainer 3
Geoffrey_Rush 3
Pedro_Velasquez 3
Jean_Brumley 3
Paula_Dobriansky 3
Johnny_Benson 3
Anthony_Mazur 3
Natasa_Micic 3
Carlos_Fasciolo 3
Qian_Qichen 3
Bernardo_Segura 3
Mike_Bair 3
Catriona_Le_May_Doan 3
```

*Figure 2: Top appearance for negative examples*

**Intersections**

Intersection might be also interesting. We noticed that 351 different people appear both on negative and positive train examples. It means that 44.5% (351/788) of the people who appear on positive examples, also appear on negative examples. We can see some of them on the following list:

```
Jean_Brumley 3
Bernardo_Segura 3
Eileen_Coparropa 3
Paul_ONeill 3
Lisa_Gottsegen 3
Carlos_Ruiz 3
Marco_Antonio_Barrera 3
Fujio_Cho 3
Ellen_DeGeneres 3
Sepp_Blatter 3
Lionel_Richie 3
Pupi_Avati 3
Wolfgang_Schuessel 3
Ted_Maher 3
Abel_Pacheco 2
Adolfo_Aguilar_Zinser 2
Gian_Marco 2
Alec_Baldwin 2
Dianne Feinstein 2
```

Figure 4: train intersection

**Validation dataset**

We will do the same analysis for validation dataset. It contains 1000 examples, 500 of them are positive and 500 are negative. 353 different people appear on positive examples, while 748 different people appear on negative examples. Let's see who appears most of the times on our positive and negative test examples:

```
Amanda_Bynes 4
John_Bolton 4
King_Abdullah_II 4
Leonid_Kuchma 4
Robert_Redford 4
Tang_Jiaxuan 4
Albert_Costa 3
Alvaro_Uribe 3
Amelie_Mauresmo 3
Ann_Veneman 3
Annette_Lu 3
Ben_Affleck 3
Bill_Frist 3
Bill_McBride 3
Bono 3
Carlos_Manuel_Pruneda 3
Denzel_Washington 3
Dick_Clark 3
Dominique_de_Villepin 3
Donald_Rumsfeld 3
```

*Figure 6: Top appearance for positive examples*

```
Damarius_Bilbo 4
Richard_Carl 4
Allison_Searing 4
Cristina_Torrens_Valero 4
Candice_Bergen 4
Chen_Shui-bian 4
Izzat_Ibrahim 3
Don_King 3
Terri_Clark 3
Amporn_Falise 3
Robert_Gordon_Card 3
Bob_Iger 3
Annie_Chaplin 3
Marc_Racicot 3
Zach_Parise 3
Lisa_Leslie 3
Gerard_Tronche 3
Bing_Crosby 3
Se_Hyuk_Joo 3
Hussam_Mohammed_Amin 3
Claire_De_Gryse 3
```

*Figure 5: Top appearance for negative examples*

**Intersections** There are 138 intersections on validation dataset. We notice that 40% (138/353) of the people who appear on positive example, also appear on negative examples. We can see some of them on the following list:

```
Candice_Bergen 4
Chen_Shui-bian 4
Elin_Nordegren 3
Pierre_Boulanger 3
Jesse_Ventura 3
Amelie_Mauresmo 2
Mary_Landrieu 2
Tippi_Hedren 2
Svetlana_Koroleva 2
Liza_Minnelli 2
Ana_Palacio 2
Annette_Lu 2
Augustin_Calleri 2
Bill_Frist 2
Donald_Pettit 2
John_Rowland 2
Brian_Cowen 2
Steve_Lavin 2
Bono 2
Melissa_Etheridge 2
Masum_Turker 2
Joe_Nichols 2
Muhammad_Ali 2
Elinor_Caplan 2
Chris_Byrd 2
```

Figure 7: validation intersection

## 9. Implementation

**Environment Constants**

root_path: full path of directory where lfwa is exported. For example. If lfwa.zip is located on /root/ the root_path will be /root/lfwa/lfw2/

images_path: inner directory, where the hierarchy is /person folder/*.jpg. the value by default is root_path + "lfw2".

TEST_CSV_PATH: full path where pairsDevTest.txt is located.

TRAIN_CSV_PATH: full path where pairsDevTrain.txt is located.

BATCH_NORM: enable/disable batch normalization

IMG_HEIGHT, IMG_WIDTH: input shape, initial dimensions for the model's input layer. Default is 105X105.

reg_lamda: the lambda value will be used for L2 regularization parameter. Default is 0.001

**Data structure**

We defined two input arrays of matrixes; each array column is a matrix – an image representation. Both input arrays on a specific index represent an example - a pair of images.
One target array represents the target values. For positive example – same person and different images, we set the target value to 1. For a negative example – 2 persons and 2 different images, we set the target value to 0.

**Functions**

- load_image(images_path, name, number, os='lnx')
  returns: image object
  description: given root directory, name and number, the function extracts a person's specific image.

- decode_img(img)
  returns: image object
  description: converts image object to array scale (division by 255) and reshape (105,105).

- load_pos_example(row)
  returns: img1, img2 pair
  description: given a row from text file, extracts a positive example and returns a pair of images of the same person.

- load_neg_example(row)
  returns: img1, img2 pair

description: given a row from text file, extracts a negative example and returns a pair of images of 2 different persons.

- make_dataset(trainrows)
  returns: first_imgs array, second_imgs array, targets array
  description: given a csv file which contains examples of pairs, the function builds the main data structure for storing dataset examples.

- lr_scheduler(epoch, lr)
  returns: LearningRate
  description: the function defines learning rate scheduler, that decay in 1% on each epoch.

- build_siamese_model()
  returns: siamese_model
  description: the function builds and prepares siamese network model.

- run_siamese_model (siamese_model, X1_All, X2_All, y_train, X1_test, X2_test, y_test )
  description: main function, runs siamese model on training dataset X1_All (images 1 array), X2_All (images 2 array) and target array (y_train). Validating on X1_test (images 1 array), X2_test (images 2 array) and target array (y_test)

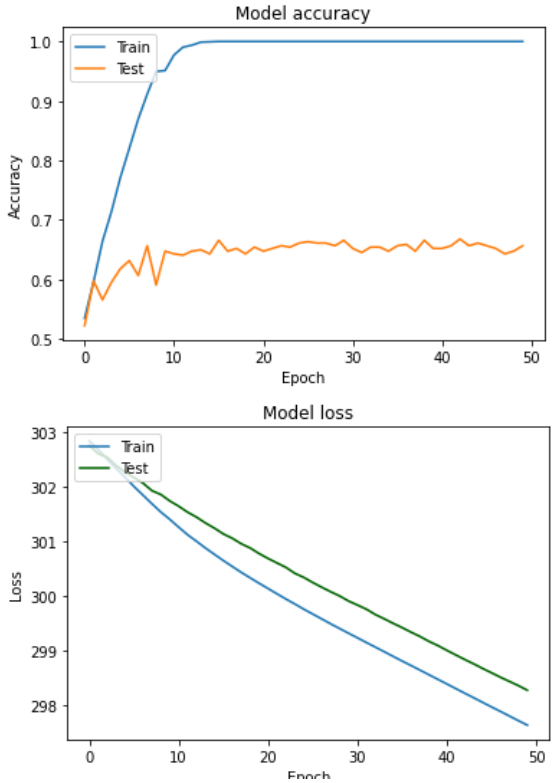
## 10. Experiments

In the following table we provide a summary of each experiment settings and results.

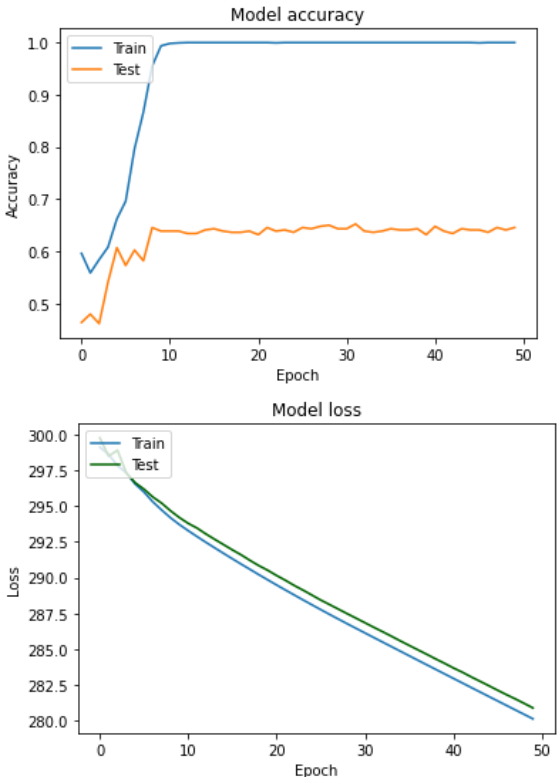
| num # | batchnorm | learning rate | optimizer | time | loss function | train loss | val loss | train ac | val acc | test loss | test acc |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | yes | 0.005 | SGD | 24ms/step | binary crossentropy | 297.63 | 298.27 | 100 | 65 | 298.33 | 61.8 |
| 2 | yes | 0.02 | SGD | 24ms/step | binary crossentropy | 280 | 280.88 | 100 | 64.5 | 280.84 | 63.9 |
| 3 | yes | 0.02 | SGD | 23ms/step | constructive loss | 263.48 | 263.63 | 100 | 56.73 | 263.62 | 55 |
| 4 | yes | 0.02 | Adam | 24ms/step | binary crossentropy | 5.49 | 5.1 | 53 | 55 | 5.08 | 59 |
| 5 | yes | 0.0002 | Adam | 24ms/step | binary crossentropy | 0.014 | 0.466 | 100 | 82.7 | 1.22 | 62.9 |
| 6 | yes | 0.001 | RMSprop | 24ms/step | binary crossentropy | 1.11 | 2.64 | 98 | 60 | 2.6 | 65 |
| 7 | yes | 0.01 | RMSprop | 24ms/step | binary crossentropy | 1.28 | 1.38 | 48 | 50 | 1.39 | 50 |
| 8 | yes | 0.0001 | RMSprop | 24ms/step | constructive loss | 113.4 | 112.4 | 100 | 73 | 112.45 | 65 |
| 9 | no | 0.02 | SGD | 2 sec 146ms/step | binary crossentropy | 284.44 | 284.98 | 100 | 69 | 258.06 | 67 |
| 10 | no | 0.0002 | Adam | 2 sec 105ms/step | binary crossentropy | 60.12 | 59.83 | 100 | 67 | 59.88 | 66.9 |
| 11 | no | 0.0002 | RMSprop | 2 sec 102ms/step | binary crossentropy | 38.11 | 37.34 | 65 | 52.9 | 37.33 | 52 |

## 11. Results

Small learning rate achieves slow improvement for SGD. However, both perform similar.

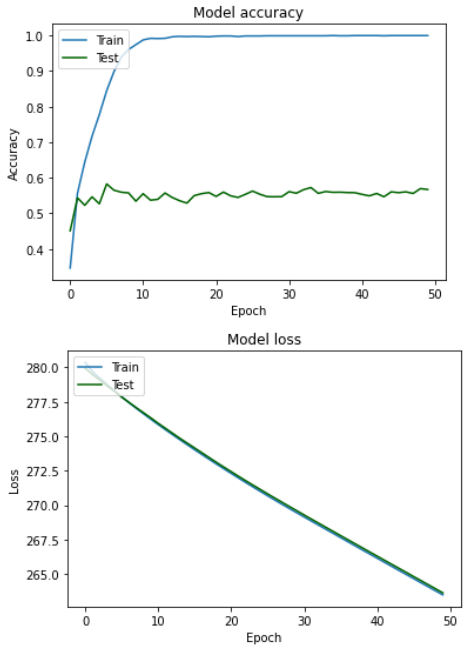



*Exp 2:SGD, lr: 0.005, loss: binary cross entropy*




*Exp 1:SGD, lr: 0.02, loss: binary cross entropy*

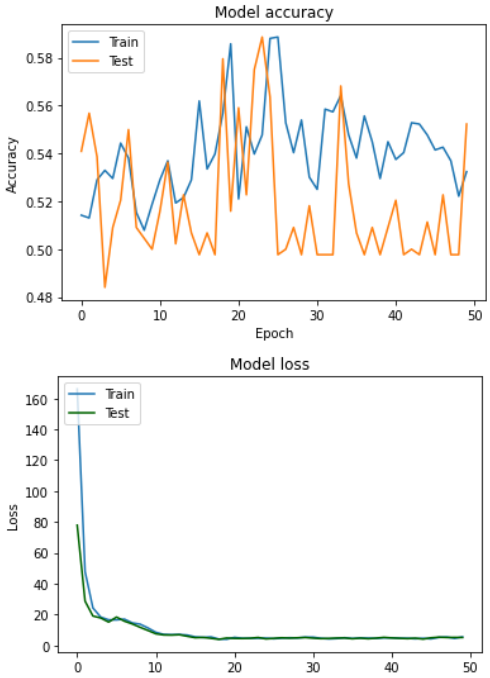
How does other loss function affect the results? We see strong over fitting on learning data. Test and validation data accuracies are close to random.



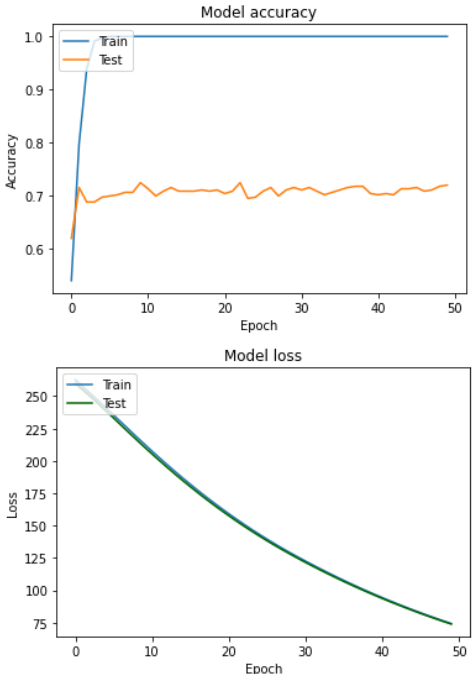

*Exp 3: SGD, lr: 0.02, loss: constructive*

We would like to examine other optimizers.

Here we see ADAM results, with different learning rates.



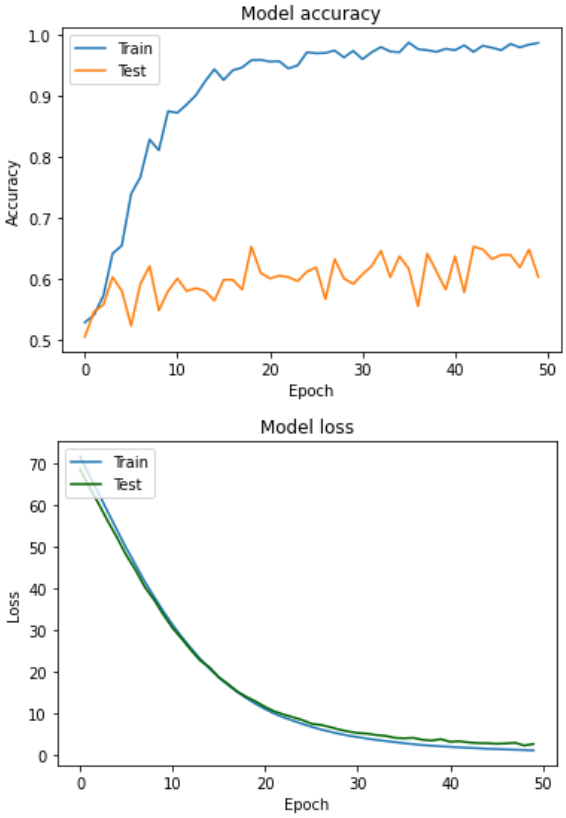

*Exp 4: Adam, lr: 0.02, loss: binary cross entropy*



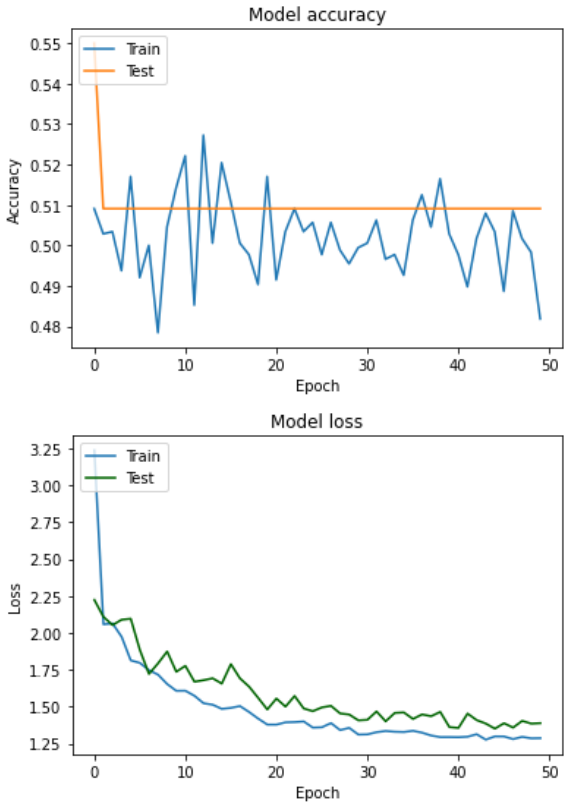

*Exp 5: Adam, lr: 0.002, loss: binary cross entropy*

These are interesting findings. ADAM requires much more smaller learning rate in order to show improvement in learning. We set its value to 100 times smaller than SGD learning rate and it looks like learning improved immediately. However, overfitting still exists.

We will now look at RMSprop optimizer.



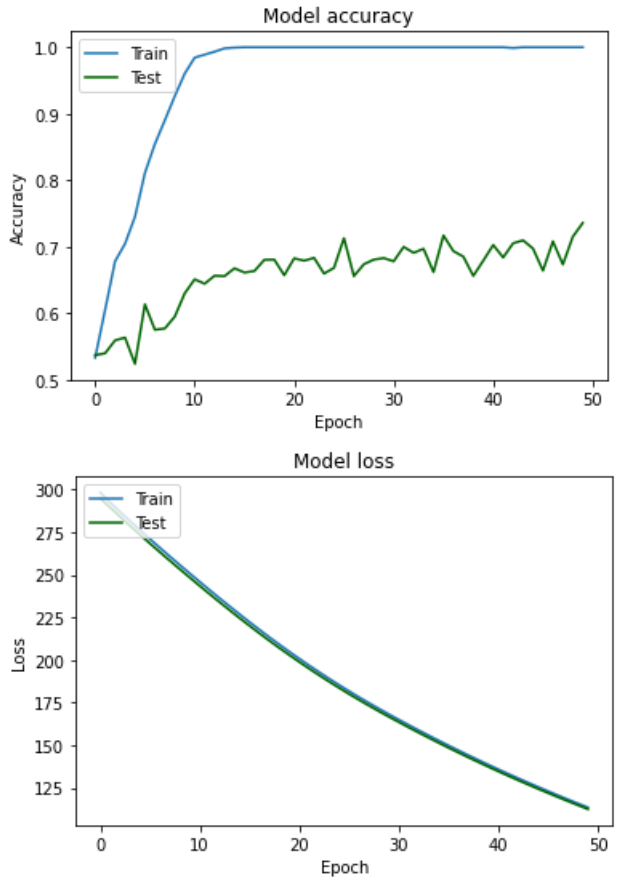

*Exp 6: RMSprop, lr: 0.001, loss: binary cross entropy*




*Exp 7: RMSprop, lr: 0.01, loss: binary cross entropy*

As seen for ADAM optimizer, RMSprop also requires very small learning rate to show improvement. The difference is very clear is this case: smaller learning rate shows improved learning.

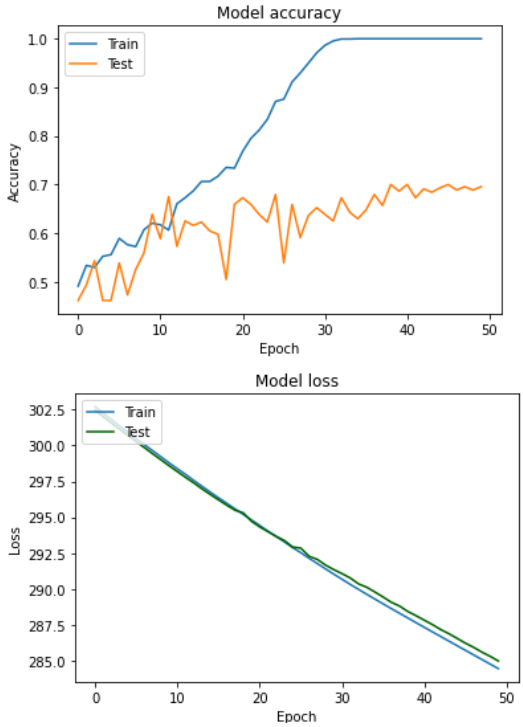
Let’s have a look for another configuration of RMSprop, where learning rate is smaller and loss function is different:



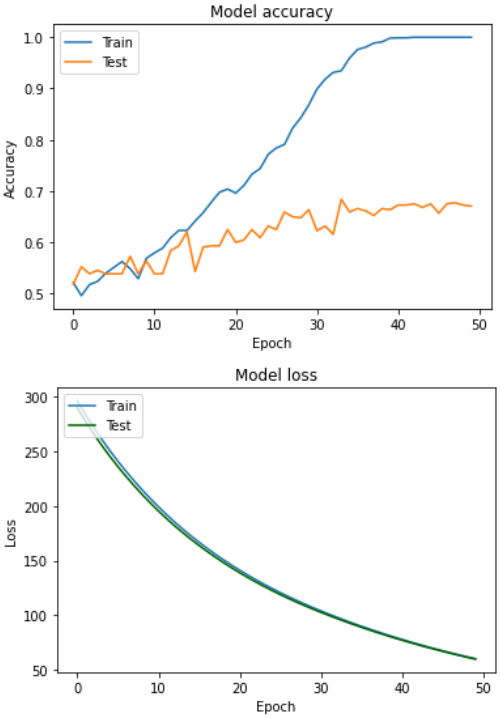

Learning has small improvement. We can see a small trend of improvement for validation set – maybe more epochs would increase accuracy.

*Exp 8: RMSprop, lr: 0.0001, loss: constructive*

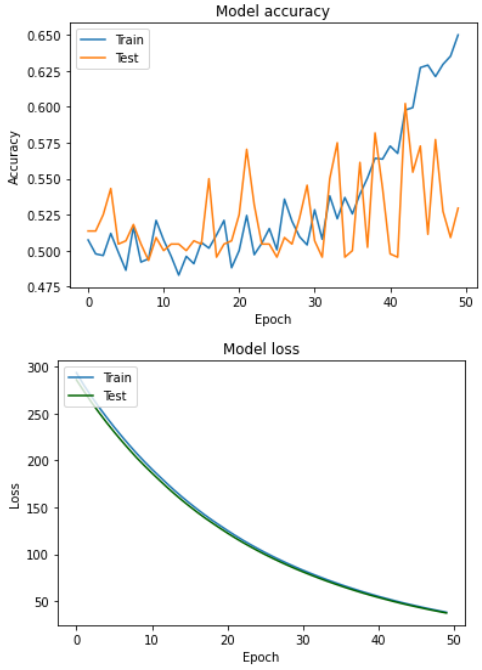
At last, we would like to examine the effect of disable performing batch normalization. Here are some examples, for different optimizers and learning rates, and without batch normalization:




*Exp 9: SGD, lr: 0.02, loss: binary cross entropy*




*Exp 10: Adam, lr: 0.0002, loss: binary cross entropy*




*Exp 11: RSprop, lr: 0.0002, loss: binary cross entropy*

**Discussion**

By examining, analyzing the results and observing the graphs, we can see that overfitting is the most noticeable characterize of the model. For every set of parameters that showed improvement on learning process – its actual accuracies for test and validation were lower, comparing to train accuracy. We also notice that different optimizers require different parameters adjustment: SGD shows best performance when learning rate is high and without batch normalization, while adaptive methods like ADAM and RMSprop require much smaller learning rate and perform best with batch normalization. Based on our experiments, we cannot determine which of the methods is recommended for one-shot learning on LFW dataset; the results are not significant. However, we still can learn a lot about combination of parameters, their mutual behavior and their correlation. We can indicate which combination perform relatively well, and it might help us for future deep learning tasks.

**Misclassification**

**Same person is recognized as different**

The following examples show us two pairs of images, where the model classified as different – but they actually belong to the same person.

The first one example belongs to the same man indeed. However, his face expression in both images is unclear: it is not a neutral or standard expression, and we assume that it might confuse the model.

The second example is more difficult to classify: closed eyes, different mustache shape (mouth is hardly seen in the right image), different background contrast - those features may justify the model classification difficulties.

**Different persons are recognized as same**

The following examples show us two pairs of images, where the model classified as the same person – but they actually belong two different people.

The first one is pretty interesting misclassification. It is obvious that they are not the same person; However, the waving hand, the large smile and bright hair – were probably recognized as strong features that indicate high similarity – and that is why the model “thinks” they are similar.

The second example is hard to justify. we could not see noticeable mutual features between the images. We assume that low brightness and dark color leads the model to determine that they are similar.

## 12. Conclusions

In this assignment we experienced implementation and configuration of siamese neural network, for performing one-shot learning. Our trained network had to decide, when given two images, whether those images belong to the same person or not.

We used LFW dataset for training and validation processes. We noticed for some limitations while working with this dataset. Overfitting of training data was our main concern; While our model showed good performance and high accuracy for training data (almost 100% accuracy in some cases) – validation and test accuracies were relatively low. We tried several methods and techniques to deal with overfitting, such as L2 Normalization with different lambda parameter values and dropout performing while building the network. Those techniques had some contribution for model performance; However, we couldn't find the best set of parameters that achieves more than 70% accuracy for validation and test data.

We believe that training over more examples would have improved model robustness. As suggested on "train, test and validation considerations" part on our report, there might be a problem when splitting given training dataset for both training and validation tasks since we cannot control overlapping of instances, which is crucial for image recognition task. We would suggest using the original dataset selection, view 1 and view 2 as described on the paper [4].

Another method for improving the model may be weights initialization with pre-trained weights, as known as "transfer learning" which shows good results for convolutions neural network training.

A final suggestion for improvement, which is commonly used for images learning tasks, is performing augmentation on the input images. the LFW data set is a known benchmark for face recognition, and it is highly qualified. However, augmentation may improve model's ability of discrimination by providing more examples of classes, that most of them, as our data analysis showed – has very small representation on the dataset.

For conclusion, we realized that one-shot learning task is challenging. While having limited training data, our model needs to "do its best" in order to recognize the main features for images discrimination, and apply its "understanding" for new unseen images; this task become more difficult when we deal with images of human faces which have their own complexity. This might explain the relatively low accuracies for validation and testing sets.
We believe that following our recommendations might be an interesting suggestion for further research.

## References

[1] Koch, Gregory, Richard Zemel, and Ruslan Salakhutdinov. "Siamese neural networks for one-shot image recognition." ICML deep learning workshop. Vol. 2. 2015.

[2] https://keras.io/examples/mnist_siamese/

[3 The Marginal Value of Adaptive Gradient Methods in Machine Learning, Ashia C. Wilson and Rebecca Roelofs and Mitchell Stern and Nathan Srebro and Benjamin Recht

[4] HUANG, Gary B., et al. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments.

[5] https://towardsdatascience.com/dropout-on-convolutional-layers-is-weird-5c6ab14f19b2

[6] http://vis-www.cs.umass.edu/lfw/pairs.txt

[7] Dimensionality Reduction by Learning an Invariant Mapping, Hadsell-et-al. 2006 http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf