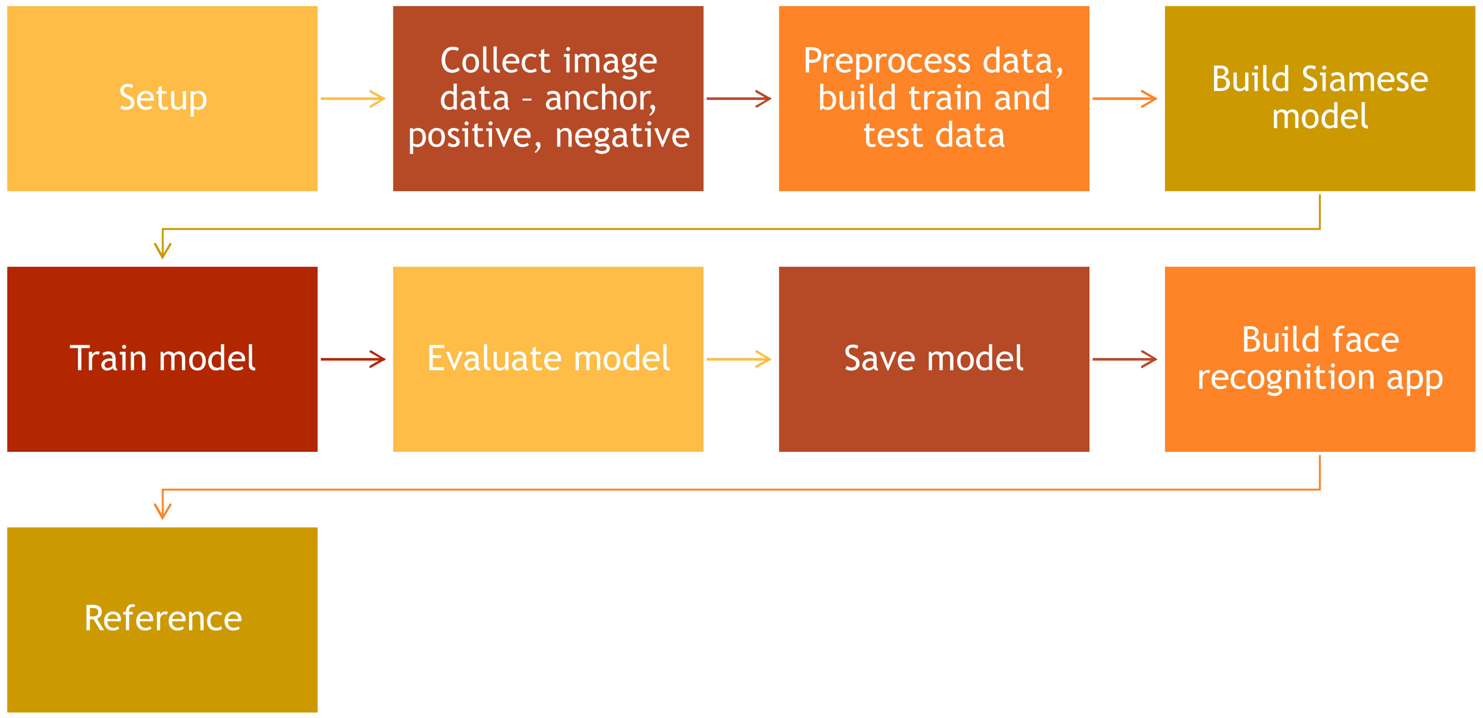
Face Recognition

Convolutional Neural Networks

Designing a lightweight and efficient Face-Recognition App

Dr. Muhammad Zeeshan

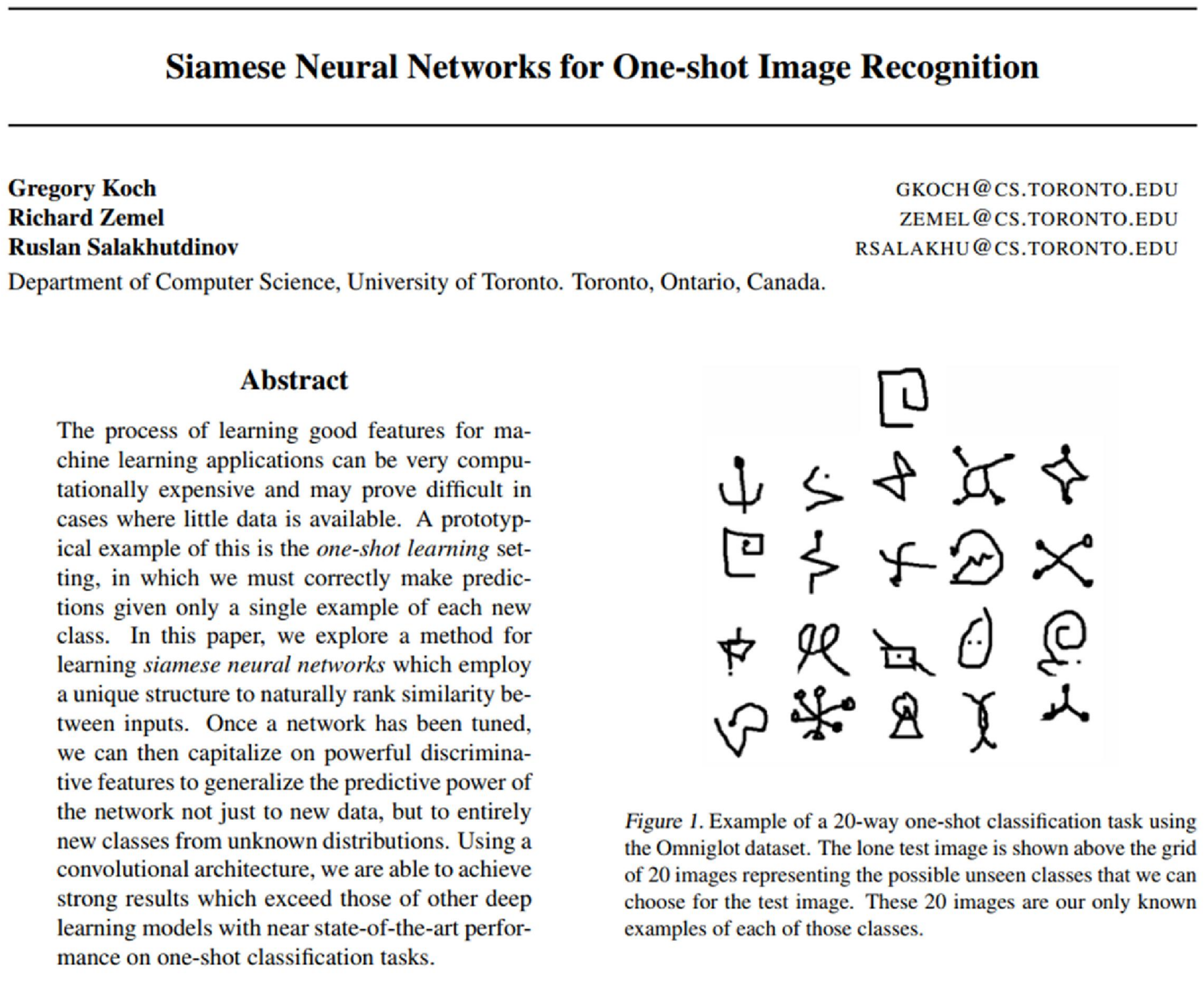
Agenda


Setup
Collect image data - anchor, positive, negative
Preprocess data, build train and test data
Build Siamese model
Train model
Evaluate model
Save model
Build face recognition app
Reference

1. Setup

- Introduction
- Siamese Neural Network
- Python standard libraries
  - os: Directory/file operation
  - random: Generate random numbers
- Python packages
  - tensorflow 2.10.0: Machine learning
  - tensorflow-gpu 2.10.0: Set GPU growth
  - opencv-python 4.6.0.66: Capture images
  - matplotlib 3.6.2: Show images for debugging
  - numpy 1.23.4: Array operation. Already installed with Python
- Set GPU growth: Better to have but not required
- Create data folder structures
  - data\anchor
  - data\positive
  - data\negative

RESEARCH PAPER



Siamese Neural Networks for One-shot Image Recognition

Gregory Koch GKOCH@CS.TORONTO.EDU
Richard Zemel ZEMEL@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science, University of Toronto. Toronto, Ontario, Canada.

Abstract

The process of learning good features for machine learning applications can be very computationally expensive and may prove difficult in cases where little data is available. A prototypical example of this is the *one-shot learning* setting, in which we must correctly make predictions given only a single example of each new class. In this paper, we explore a method for learning *siamese neural networks* which employ a unique structure to naturally rank similarity between inputs. Once a network has been tuned, we can then capitalize on powerful discriminative features to generalize the predictive power of the network not just to new data, but to entirely new classes from unknown distributions. Using a convolutional architecture, we are able to achieve strong results which exceed those of other deep learning models with near state-of-the-art performance on one-shot classification tasks.

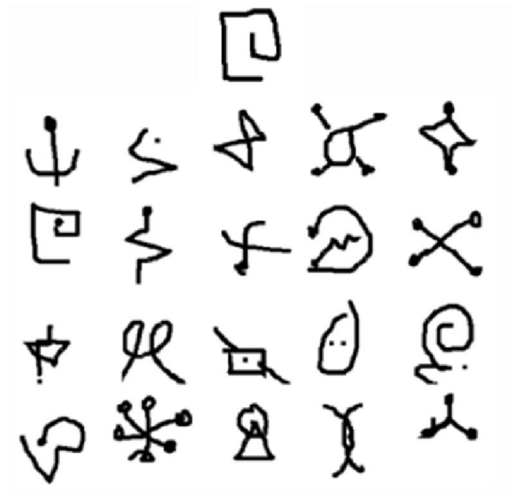


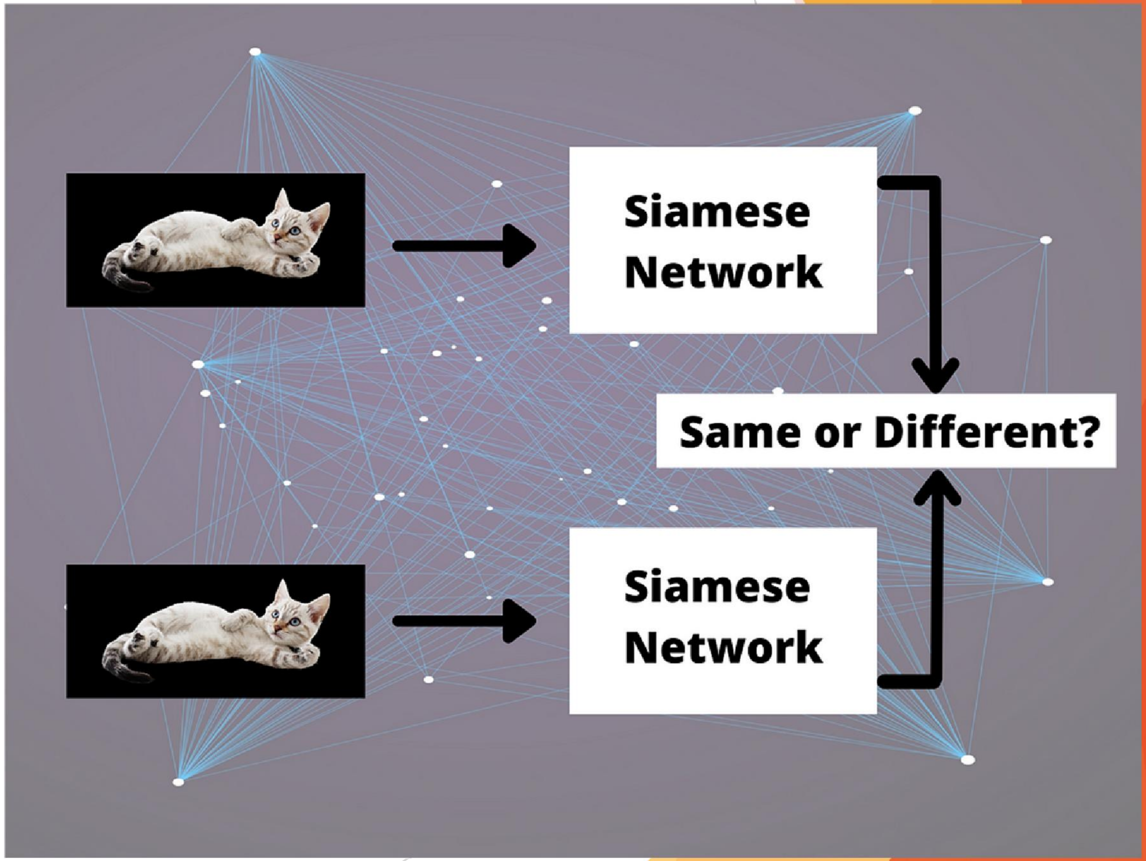
Figure 1. Example of a 20-way one-shot classification task using the Omniglot dataset. The lone test image is shown above the grid of 20 images representing the possible unseen classes that we can choose for the test image. These 20 images are our only known examples of each of those classes.

Neural Network for Image Recognition

- “Siamese Neural Networks for One-shot Image Recognition”
  - Gregory Koch, Richard Zemel, Ruslan Salakhutdinov
  - Department of Computer Science, University of Toronto, Ontario, Canada
- A Siamese Neural Network (SNN) is a class of neural network architectures that contain two or more identical sub-networks
  - “Identical” – they have the same configuration with the same parameters and weights
  - Parameter updating is mirrored across both sub-networks and it’s used to find similarities between inputs by comparing its feature vectors

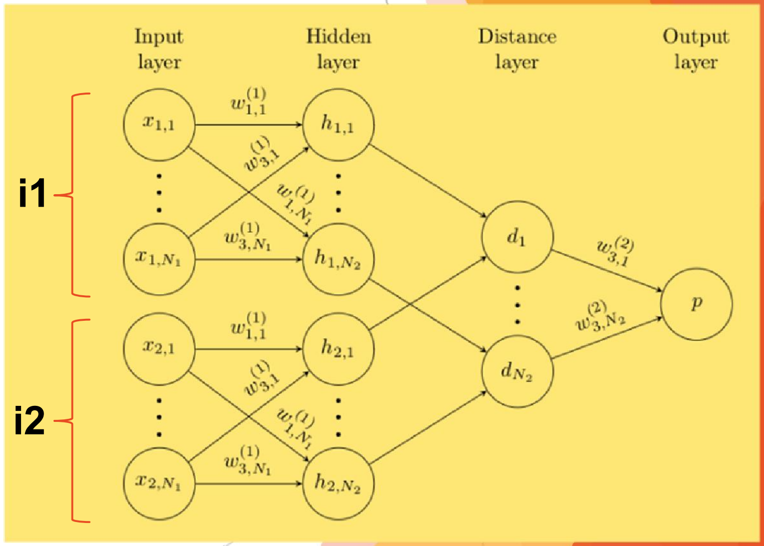
Siamese Neural Network (SNN)

- Siamese neural networks are composed of 2 identical subnetworks that output 2 **embeddings** (**vector representation** of input)
- These embeddings are then used as inputs to a loss function.
- This loss function is designed to minimize the distance between similar inputs (2 images of 2 faces that belong to the same person) and maximize the distance between dissimilar inputs (2 faces of 2 different people)



Siamese Neural Network (SNN)

- We pass two images to SNN, compute distance vector between features to figure out whether they are same or not
- Distance layer measures the similarity between the two images
  - Similar - output is 1
  - No similar - output is 0



One-shot Learning

- Typically, classification involves fitting a model given many examples of each class, then using the fit model to make predictions on many examples of each class
- One-shot learning is an ML-based object classification algorithm that assesses the similarity and difference between two images
  - One-shot learning is a classification task where one example (or a very small number of examples) is given for each class, that is used to prepare a model, that in turn must make predictions about many unknown examples in the future
  - One-shot learning for face recognition is achieved by learning a rich low-dimensional feature representation, called a **face embedding**
  - Embeddings were learned for one-shot learning problems using a **Siamese network**

The Process

1. Setup
2. Collect image data – anchor, positive, negative
3. Preprocess data, build train and test data
4. Build Siamese model
5. Train model
6. Evaluate model
7. Save model
8. Build face recognition app
9. Reference

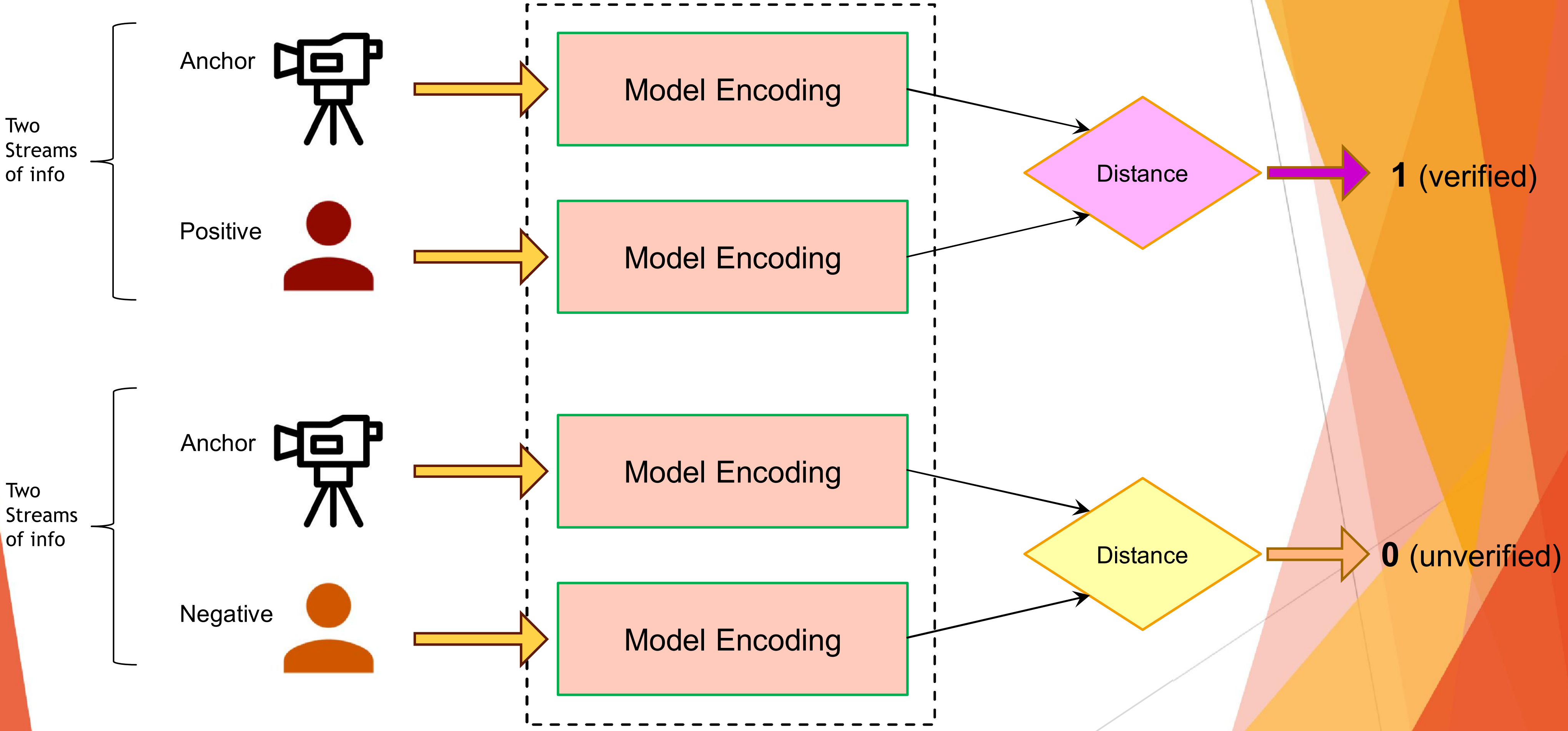
Import dependencies, build image datasets (anchor, positive, negative)

1. Setup

- Python 3.10.8
- Python standard libraries
  - os: Directory/file operation
  - random: Generate random numbers
- Python packages
  - tensorflow 2.10.0: Machine learning
  - tensorflow-gpu 2.10.0: Set GPU growth
    - Supports GPU accelerated operations via Nvidia CUDA
  - opencv-python 4.6.0.66: Capture images
  - matplotlib 3.6.2: Show images for debugging
  - numpy 1.23.4: Array operation. Already installed with Python
- Set GPU growth: Better to have but not required
- Create data folder structures
  - data\anchor
  - data\positive
  - data\negative

2. Collect image data – anchor, positive, negative

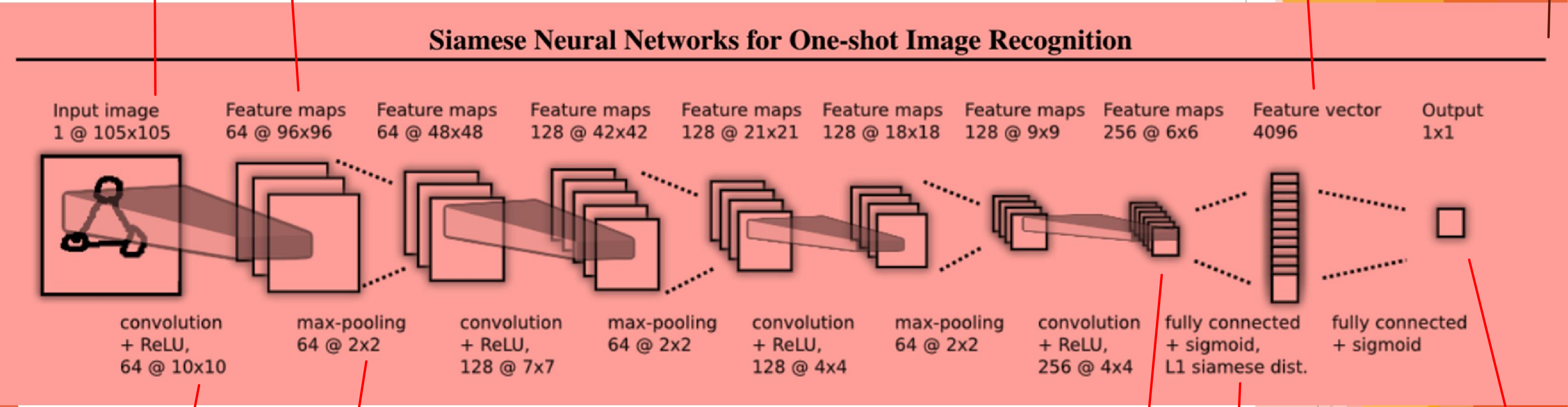
- data\anchor (input image, captured from camera)
  - As input images, to compare with positive/negative images
  - Collect 300+ images via camera using OpenCV
  - Adjust brightness/contrast/saturation/etc. to augment to 3000
- data\positive
  - The target images for input image to compare with
  - Collect 300+ images via camera using OpenCV
  - Adjust brightness/contrast/saturation/etc. to augment to 3000
- data\negative
  - The negative images for input image to compare with
  - Download from http://vis-www.cs.umass.edu/lfw/
- Image dimensions: 250 x 250 pixel


Two Streams of info
Anchor
Positive
Model Encoding
Model Encoding
Distance
1 (verified)
Two Streams of info
Anchor
Negative
Model Encoding
Model Encoding
Distance
0 (unverified)

64 channels / feature maps, 96x96 each

Input image 105x105

Input image (anchor and +ve / -ve image) has been transformed into a feature vector



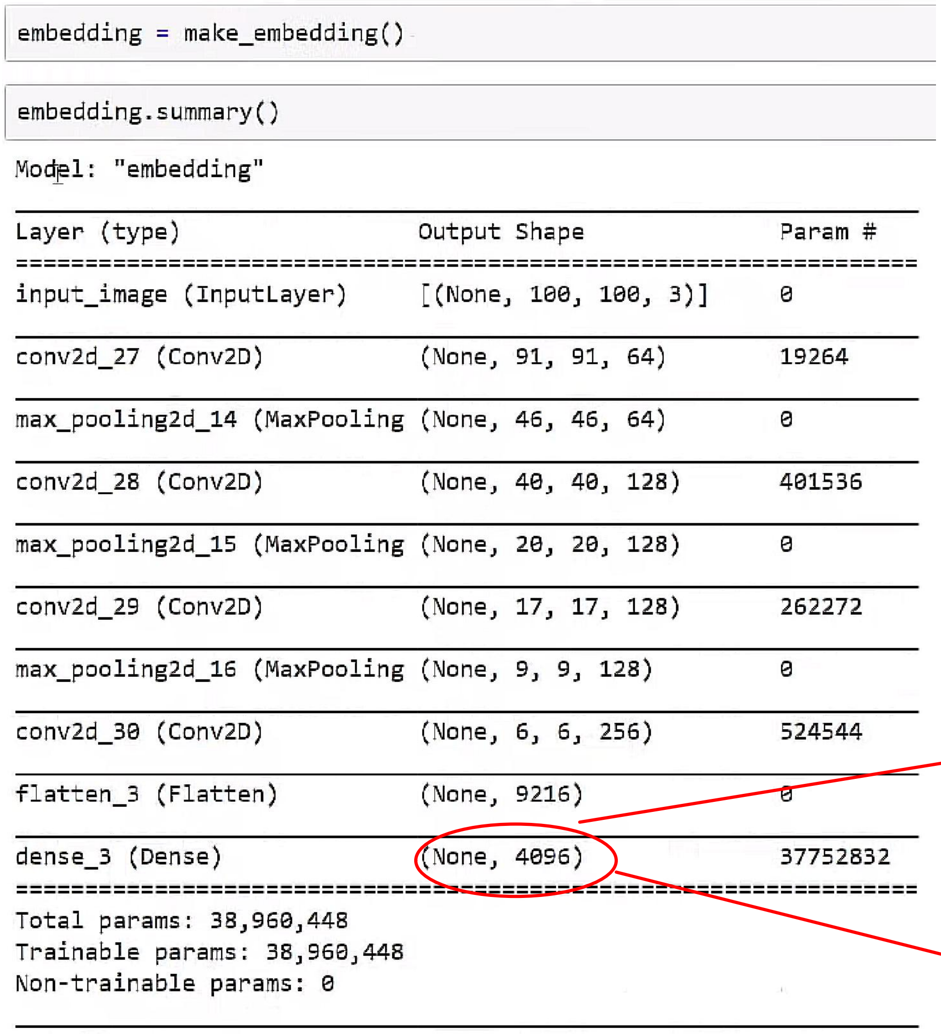
Flatten Layer
Input: 256 @ 6x6
Output: 256x6x6=9216

Decision whether 2 images are similar or different

64 filters of 10x10 each, stride 1

64 filters of 2x2 each

Dense Layer
Input: 9216
Output: 4096



```
embedding = make_embedding()
```

```
embedding.summary()
```

```
Model: "embedding"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_image (InputLayer)    [(None, 100, 100, 3)]     0
_________________________________________________________________
conv2d_27 (Conv2D)          (None, 91, 91, 64)        19264
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 46, 46, 64)       0
_________________________________________________________________
conv2d_28 (Conv2D)          (None, 40, 40, 128)       401536
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 20, 20, 128)      0
_________________________________________________________________
conv2d_29 (Conv2D)          (None, 17, 17, 128)       262272
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 9, 9, 128)        0
_________________________________________________________________
conv2d_30 (Conv2D)          (None, 6, 6, 256)         524544
_________________________________________________________________
flatten_3 (Flatten)         (None, 9216)              0
_________________________________________________________________
dense_3 (Dense)             (None, 4096)              37752832
=================================================================
Total params: 38,960,448
Trainable params: 38,960,448
Non-trainable params: 0
_________________________________________________________________
```

Output of model or embedding layer

This represents output of one data stream (anchor/ +ve / -ve)

Preprocess Images, Train and Test partitions, Siamese Model, Training, Evaluation

Import tensorflow Dependencies

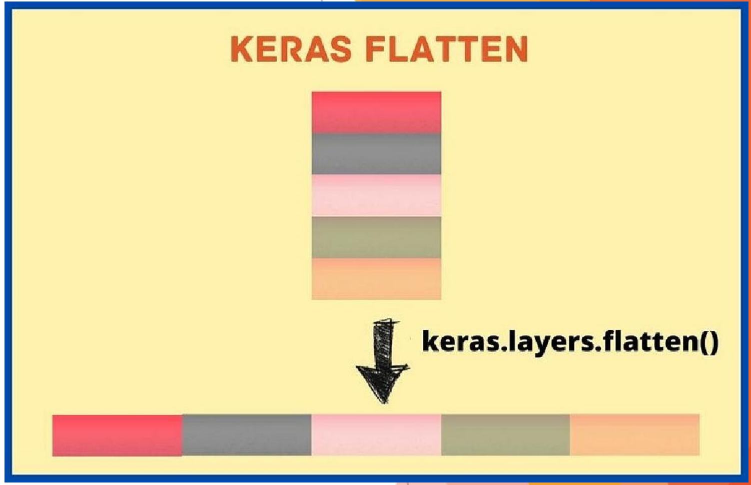
- from tensorflow.keras.models import Model
  - class Model: A model grouping layers into an object with training/inference features
- from tensorflow.keras.layers import Layer, Conv2D, Dense, MaxPooling2D, Input, Flatten
  - Layer: Keras layers API
  - Creates custom layers
  - https://keras.io/api/layers/
  - Conv2D: 2D convolution layer (e.g. spatial convolution over images)
  - This layer creates a **convolution kernel** that is convolved with the **layer input** to produce a tensor of outputs
  - If **use_bias** is **True**, a bias vector is created and **added** to the outputs
  - If **activation** is not None, it is **applied to the outputs** as well
  - Dense:
  - Densely-connected NN layer
  - Dense implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is True). These are all attributes of Dense
  - https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

Import tensorflow dependencies

- from tensorflow.keras.layers import Layer, Conv2D, Dense, MaxPooling2D, Input, Flatten
  - MaxPooling2D:
  - Max pooling operation for 2D spatial data
  - Downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window (of size defined by pool_size) for each channel of the input
  - The window is shifted by strides along each dimension
  - Input:
  - Layer to be used as an entry point into a Network
  - Input() is used to instantiate a ***Keras tensor***
  - A Keras tensor is a tensor object from the underlying backend (Theano or TensorFlow), which we augment with certain attributes that allow us to build a Keras model just by knowing the inputs and outputs of the model
  - In TensorFlow, a tensor is a multi-dimensional array. It is an essential data structure used for building and training deep learning models. Tensors can be of different types such as float, int, string, etc

Import tensorflow dependencies

- from tensorflow.keras.layers import Layer, Conv2D, Dense, MaxPooling2D, Input, Flatten
  - Flatten:
  - Flattens the input. Does not affect the batch size
  - Converts matrices into vectors



- from Utils import POS_PATH, NEG_PATH, ANC_PATH, preprocess, L1Dist, SIAMESE_Model_NAME
  - Python Utils is a collection of small Python functions and classes which make common patterns shorter and easier
  - It is by no means a complete collection but it has served me quite a bit in the past and I will keep extending it

Load and Preprocess Images

- tf.data
  - The tf.data API is used to build complex input pipelines
  - For example, the pipeline for an image model might aggregate data from files in a distributed file system, apply random perturbations to each image, and merge randomly selected images into a batch for training
- tf.data.Dataset
  - The tf.data API introduces a tf.data.Dataset abstraction that represents a sequence of elements, in which each element consists of one or more components
  - For example, in an image pipeline, an element might be a single training example, with a pair of tensor components representing the image and its label
- tf.data.Dataset.list_files()
  - Used to create a dataset of all files matching a pattern
- tf.io.read_file()
  - Reads the contents of file.
- tf.io.decode_jpeg()
  - Decode a JPEG-encoded image to a uint8 tensor.

Load and Preprocess Images

- tf.data.Dataset.zip()
  - The **tf.data.zip()** function is used for creating a dataset by zipping together a dict, array, or nested structure of Dataset
- data.map()
  - TensorFlow map() method of tf.data.Dataset is used to perform different preprocessing operations e.g., transformationg, normalization etc.
- data.shuffle()
  - rearrange/muddle images so that positive and negative images are mixed up
  - Shuffling is important since we need to have a mixed set of samples in training and testing partitions

3. Preprocess data, build train and test data

- tf.data: Build TensorFlow input pipelines
  - Loop through a specified directory and grab images
- Preprocessing
  - Resize image to be 100 x 100 x 3
  - Scale image to be between 0 and 1
- Create labelled dataset
  - (anchor, positive, 1.0): anchor compares with positive, output as 1.0/true
  - (anchor, negative, 0.0): anchor compares with negative, output as 0.0/false
- Build train/test data
  - Shuffle dataset
  - Train data: 70%
  - Test data: 30%

Embedding Layers

- An embedding layer is a type of hidden layer in a neural network
  - It maps input information from a high-dimensional to a lower-dimensional space
  - It allow the network to learn more about the relationship between inputs and to process the data more efficiently
- Feature mapping pipeline
  - Embedding layer is the first layer that processess images
  - It converts raw images into a data representation, the data that passes through a siamese NN
  - Embedding layer in a CNN forms a feature mapping pipeline
- Types of embedding layers:
  - Embedding layer type depends on the NN and the embedding process
  - Text embedding
  - Image embedding
  - Graph embedding and others

Image Embedding

- Image embedding is a technique for representing images as dense embedding vectors
- These vectors capture some visual features of the image, and we can use them for tasks such as image classification, object detection, and similar
- Popular pre-trained CNN that can be used to generate image embeddings
  - NFNets, EfficientNets, ResNets

4. Build Siamese model

- Build embedding layer
  - Input(100,100,3)
  - Convolution + ReLU, 64 @ 10 x 10
  - Max-pooling, 64 @ 2 x 2
  - Convolution + ReLU, 128 @ 7 x 7
  - Max-pooling, 64 @ 2 x 2
  - Convolution + ReLU, 128 @ 4 x 4
  - Max-pooling, 64 @ 2 x 2
  - Convolution + ReLU, 256 @ 4 x 4
  - Fully connected + sigmoid

- Build distance layer (Siamese L1 distance)
  - **Siameses L1 Distance** layer compares '**input_embedding**' and '**validation_embedding'** by taking their difference
  - It performs similarity measure between:
    1. anchor and +ve images
    2. anchor and –ve images
- Make Siamese model
  - Embedding (input_image)
  - Embedding (validation_image)
  - Similarity calculation via distance layer

5. Train model

- Setup loss and optimizer
  - NN tries to minimize loss
  - Optimizer performs backpropagation through NN
- tf.losses.BinaryCrossentropy()
- tf.keras.optimizers.Adam(1e-4) # 0.0001
- Establish checkpoints
  - Placeholders to reload NN from

5. Train model

- Build train step – defines the processes that take place during training on a single batch
  - Get train data: (anchor, positive/negative, 1.0/0.0)
  - Forward pass: Get Siamese model output / prediction
  - Calculate loss (binary cross entropy)
  - Calculate gradients of weights across the NN
  - Optimizer back propagate through NN
  - Calculate updated weights and apply to Siamese model

5. Train model

- Build train loop
  - Training step is applied over a single batch
  - Training loop applies training step over entire dataset
- Train model

9. Reference

Tutorial:
https://www.youtube.com/watch?v=LKispFFQ5GU

Source code:
https://github.com/nicknochnack/FaceRecognition

Paper: Siamese Neural Networks for One-shot Image Recognition