Member-only story

Siamese Neural Networks with Triplet Loss and Cosine Distance

Theory & Code-along: Triplet loss with cosine distance for Siamese Networks on CIFAR-10 dataset

Tan Pengshi Alvin · Follow

Published in Towards Data Science
11 min read · May 12

Listen Share More

Image by Alex Meier on Unsplash

What if we could encode every object image (human faces, etc) into a template — a vector of numbers? Thereafter, we could objectively determine the similarity between objects by numerically comparing — finding the distances — between their templates. In deep learning, this is exactly what the Siamese Neural Networks hope to achieve.

The Siamese Neural Networks are basically models that, upon training, generate distinct feature vectors (templates) for each input object. Although these models are typically adopted in templating object images (Computer Vision) generally speaking, they may also be employed for text and sound data.

Apart from security authentication, such as facial recognition and signature comparisons, Siamese Neural Networks are also commonly used in E-Commerce platforms to measure product similarity. For instance, some E-Commerce platforms allow you can search for similar products by uploading an image of an object you are looking for. On Kaggle, there is even a Product Matching competition by Shopee, a leading E-Commerce company in South-East Asia.

In this article, we will explore a common data set in Tensorflow — the CIFAR-10 — that bears some semblance to the problem of product similarity search, except that the objects of interest are automobiles — cars, planes, trucks, ships, etc — and animals (or pets if you like!) — cats, dogs, horses, birds, deers, etc.

Before we begin, we have to first understand the theory behind Siamese Neural Network. Thereafter we will explore the codes for training and evaluating a simple Siamese Neural Network on the CIFAR-10 data set.

Ready? Let's start!

. . .

1. Theory of Siamese Networks

I have to admit the cover image in this article is a little misleading — the word 'Siamese' in fact does not come from the 'Siamese Cat'. Rather it comes from the

'Siamese Twins', or conjoined twins — twins who are conjoined in one part of the body.

Hence, the Siamese Neural Networks basically mean twin neural networks, that are typically conjoined at the end — the Lambda layer, as we will see — before feeding the model output to the loss function. During the training of these twin networks, their weights are exactly the same between them during the initialization, forward propagation and backpropagation process.

Because we are generally working with images, each body of twin neural networks is typically a Convolutional Neural Network (CNN). If you are unfamiliar with CNN or need to refresh your idea, I have an excellent article about it here:

Transfer Learning and Convolutional Neural Networks (CNN)

A complete guide from CNN to Transfer Learning for Kaggle's Cat versus Dog dataset

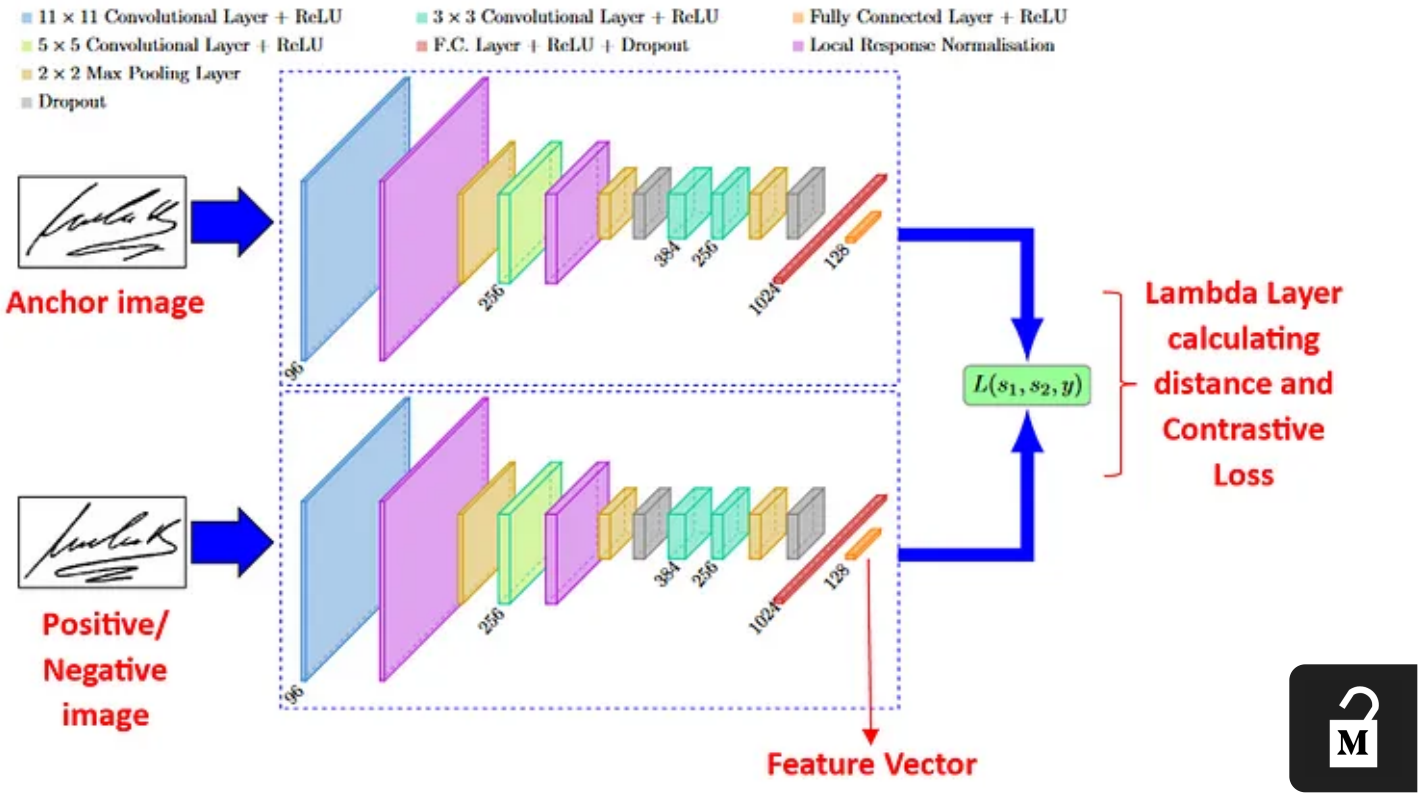
medium.com

With this idea in mind, we will introduce 2 common types of Siamese Neural Networks:

1.1 Contrastive Loss Siamese Networks

The first type is the Siamese Neural Networks based on calculating the Euclidean/Cosine distance between the embedding layers — the feature vectors — of twin CNNs, before comparing with the ground truths (1:Match, 0:Non-Match) to determine the Contrastive Loss.

Below is an illustration of such a model:



Example of Contrastive Loss Siamese Neural Networks. Image adapted from the SigNet paper.

$$L = (1 - Y) * \|x_i - x_j\|^2 + Y * \max(0, margin - \|x_i - x_j\|^2)$$

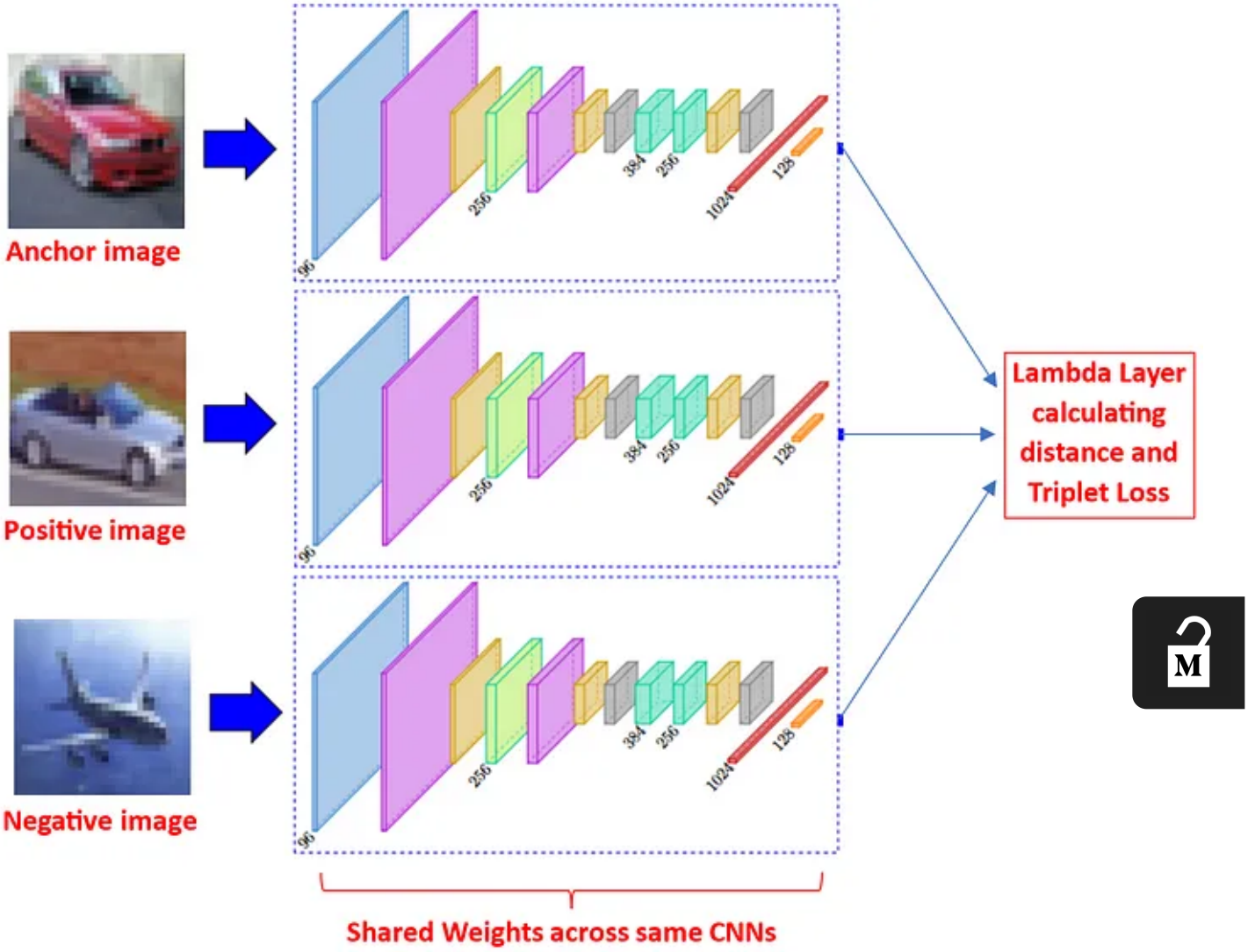
Contrastive Loss formula with Euclidean Distance, where Y is the ground truth. Image by Author.

1.2 Triplet Loss Siamese Networks

The second type of Siamese Neural Networks is based on calculating the 2 Euclidean/Cosine distances among the embedding layers (feature vectors) — between the Anchor and Positive Image, and between the Anchor and Negative Image — of triplet CNNs, and then computing the Triplet Loss completely in the Lambda layer, without comparing with any ground truths.

Because research has shown that this Triplet Loss model is generally more robust than the Contrastive Loss model, we will focus on this type of Siamese Network in this article.

Below is an illustration of such a model:



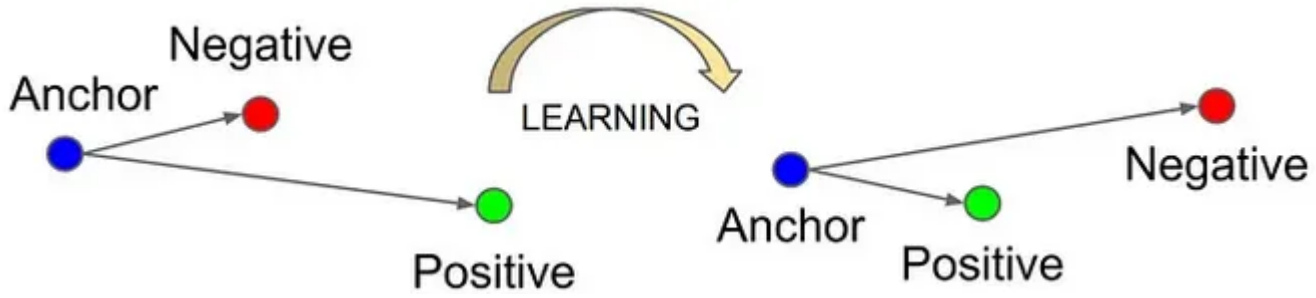
Example of Triplet Loss Siamese Neural Networks. Image adapted from the SigNet paper.

$$L(A, P, N) = \max\left(\|f(A) - f(P)\|_2 - \|f(A) - f(N)\|_2 + margin, 0\right)$$

Triplet Loss formula with Euclidean Distance, where A is the Anchor image input, P is the Positive image input and N is the Negative image input. Image by Author.

1.3 The Goal of Siamese Networks

Now, we have seen the approximate architectures of the Siamese Neural Networks. But upon the training of the networks, what are we intending to achieve here? Let's take a look at the illustration below:



Training of Siamese Networks reduces distances between similar images while increasing distance between dissimilar images. Image by the FaceNet paper.

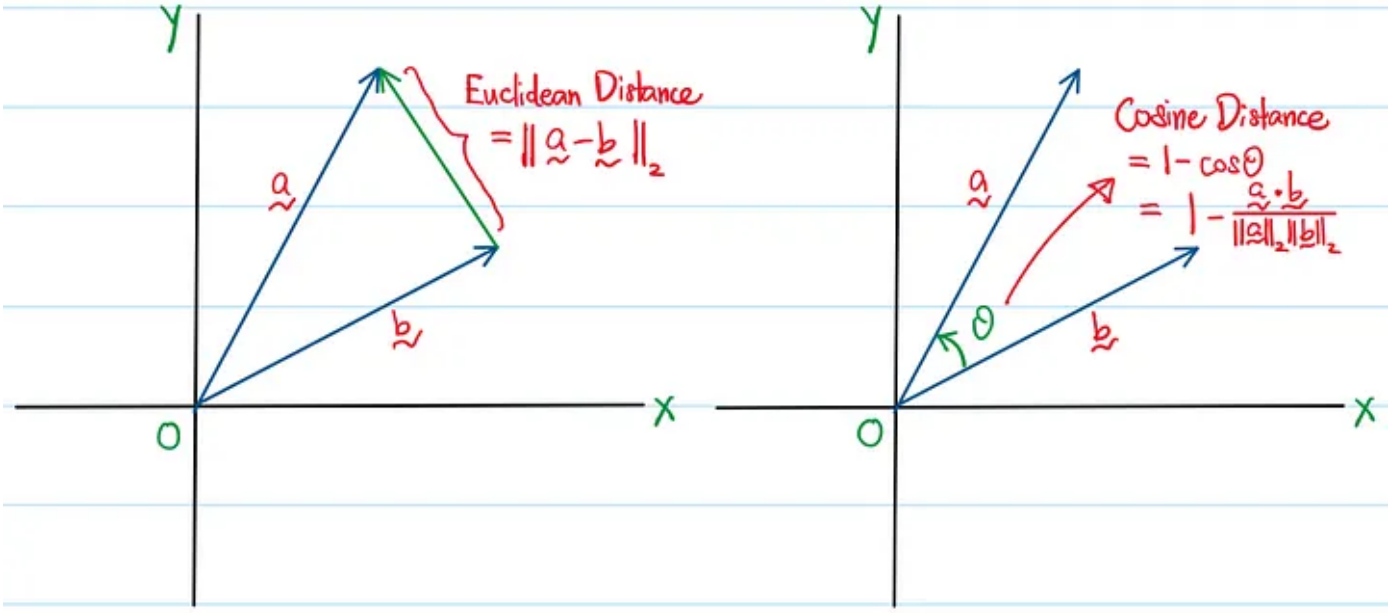
We see that the Siamese Networks are learning to recreate similar feature vectors between images of the same class. As such, after training, the distances between similar images' templates will decrease, while that between dissimilar images' templates will increase.

With that being said, it is important to cover as many classes of images as possible during training, such that the model may also generalize to unseen classes (signatures, faces, etc).

Finally, during model evaluation, we are chiefly concerned with generating templates of input image data. Hence, only a single CNN network or body of the twins/triplet networks is extracted, without the Lambda layer, when running template inference.

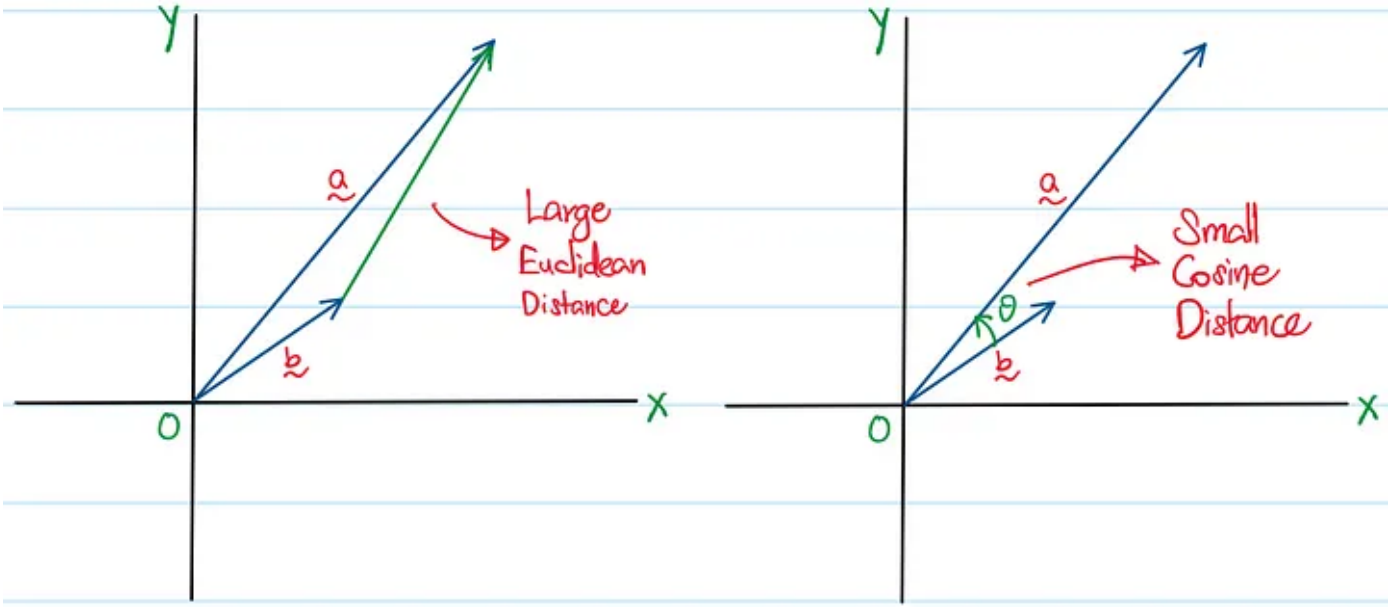
1.4 Euclidean Distance versus Cosine Distance

Before we move to coding, let us first differentiate between 2 common vector distance metric — the Euclidean distance and the Cosine Distance. So far in the above illustration, we have shown the Euclidean distance, because it is more intuitive to understand, but not necessarily more superior to the Cosine distance in building a better model. Let's illustrate below:



The illustration of the Euclidean Distance and Cosine Distance between two vectors in a 2-D space. Image by Author.

From the above, it is clear that the Euclidean distance is simply the 'coordinate distance' between 2 feature vectors, while the Cosine distance is one measure of the 'angular distance' between them. Hence, when 2 feature vectors are oriented far apart, we can see both Euclidean distance and Cosine distance are similarly huge. But there is a subtle difference and let's see below:



A comparison between Euclidean Distance and Cosine distance at small angles but with difference in vector length. Image by Author.

While the Cosine distance measures only the angular difference between feature vectors, the Euclidean distance measures a second dimension — length difference. As such, while more intuitive, the Euclidean distance is inherently a more complex metric than the Cosine distance.

Generally speaking, both the Euclidean and Cosine distance are widely used, and the choice largely depends on empirical exploration. Nonetheless, for a smaller data set and dedicated number of classes, adopting the Cosine distance in the loss function could be a better choice, and that is what we would be doing for our CIFAR-10 data set.

2. Siamese Networks Code-Along

Next, let's get our hands dirty with coding. We will adopt engineering the Triplet Loss Siamese Networks on the TensorFlow CIFAR-10 data set. We will base our Triplet Loss on the Cosine Distance, and then during the evaluation of test set, compare test im using Angular Similarity.

***Note**: Angular Similarity is used, as it is based on Cosine Distance, except that the values are scaled between 0% to 100%.*

$$similarity_{angular} = 1 - \frac{\arccos\left(\frac{A \bullet B}{\|A\|\|B\|}\right)}{\pi}$$

The formula for Angular Similarity. Image by Author.

Also note that in the model initialization process, we would be adopting the TensorFlow Functional API (contrast with the Sequential API we used in the Transfer Learning and CNN article introduced earlier), together custom Lambda layer and a custom loss function.

Without further hesitation, let's dive into coding!

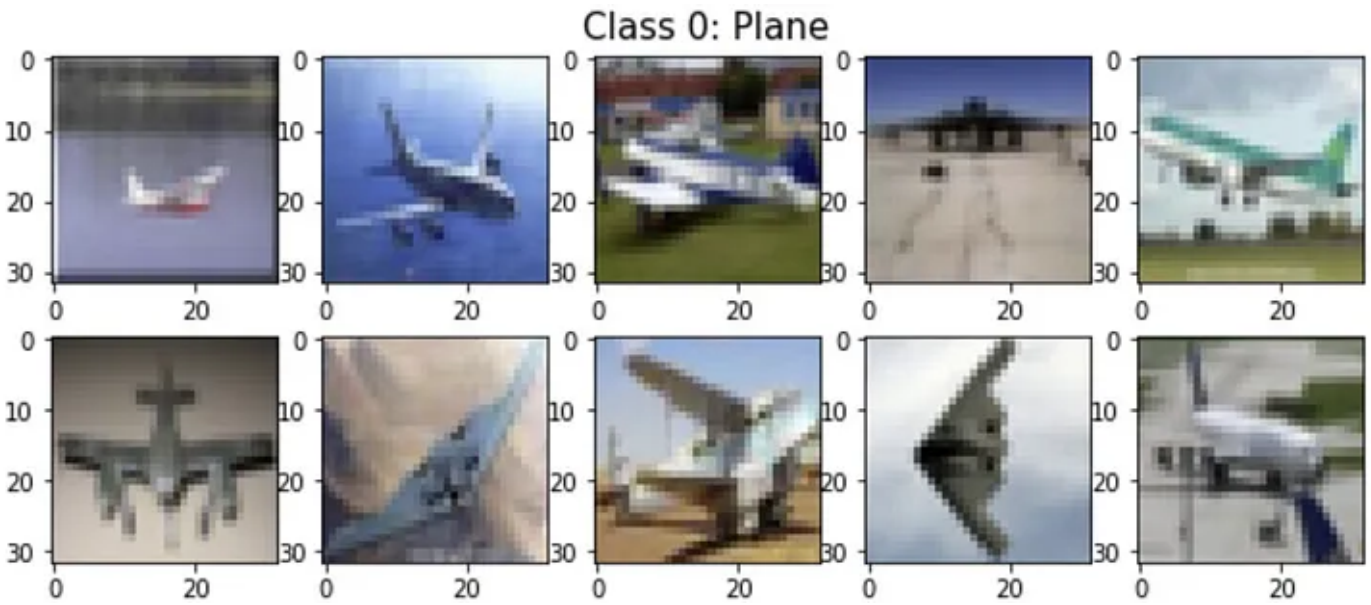
. . .

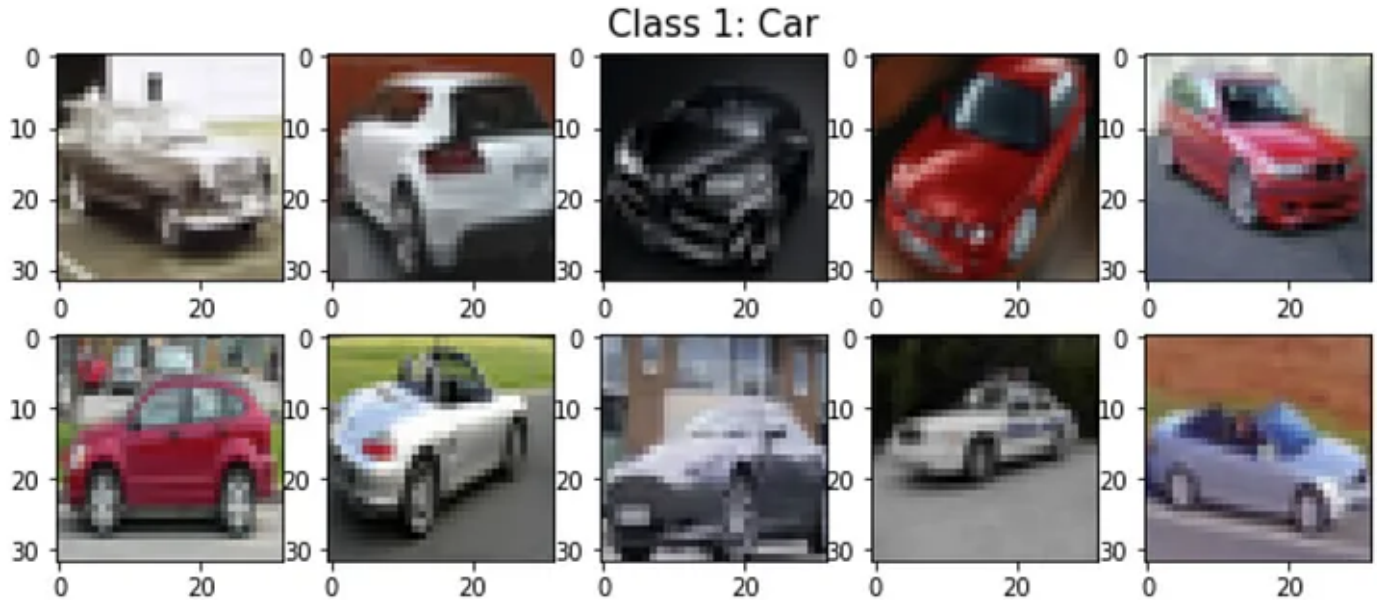
2.1 Exploring the CIFAR-10 Data Set

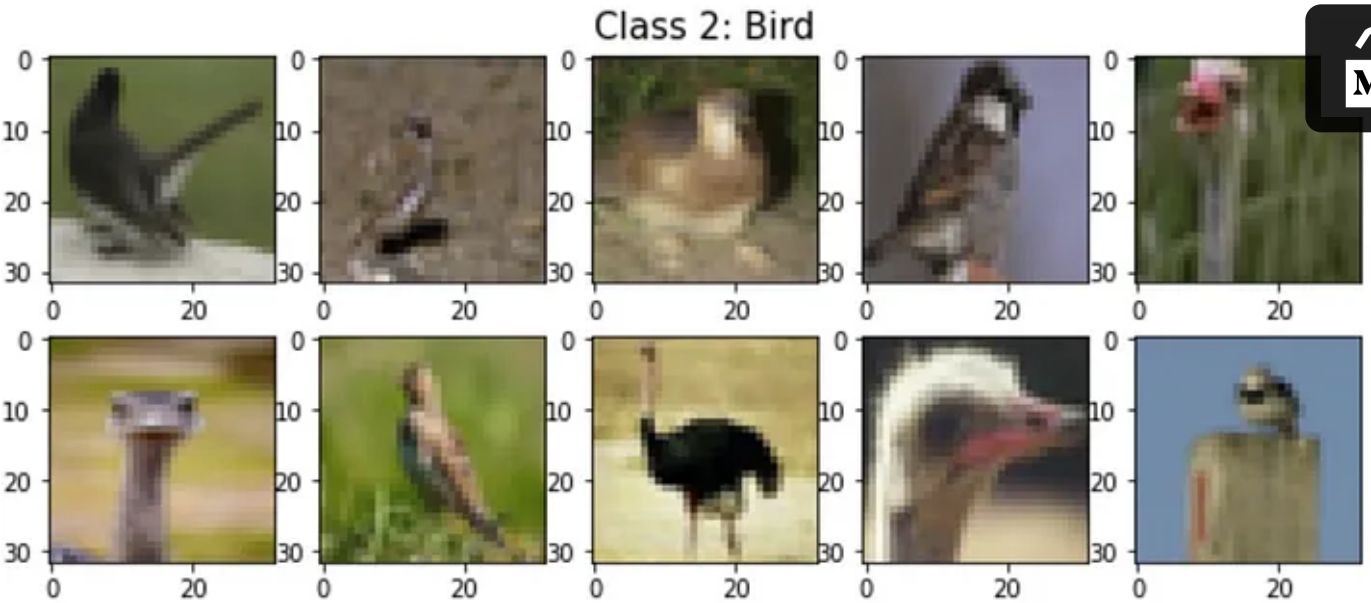
```
# import necessary libraries
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

# set random seed
np.random.seed(42)

# load CIFAR-10 data
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar10.load_data()

# check data size
assert X_train.shape == (50000, 32, 32, 3)
assert X_test.shape == (10000, 32, 32, 3)
assert y_train.shape == (50000, 1)
assert y_test.shape == (10000, 1)

# combine data first - we will generate test set later.
X = np.concatenate([X_train,X_test],axis=0)
y = np.concatenate([y_train,y_test],axis=0)
y = np.squeeze(y)

assert X.shape == (60000, 32, 32, 3)
assert y.shape == (60000,)

# check number of data in each class
unique, counts = np.unique(y,return_counts=True)
np.asarray([unique,counts]).T
```

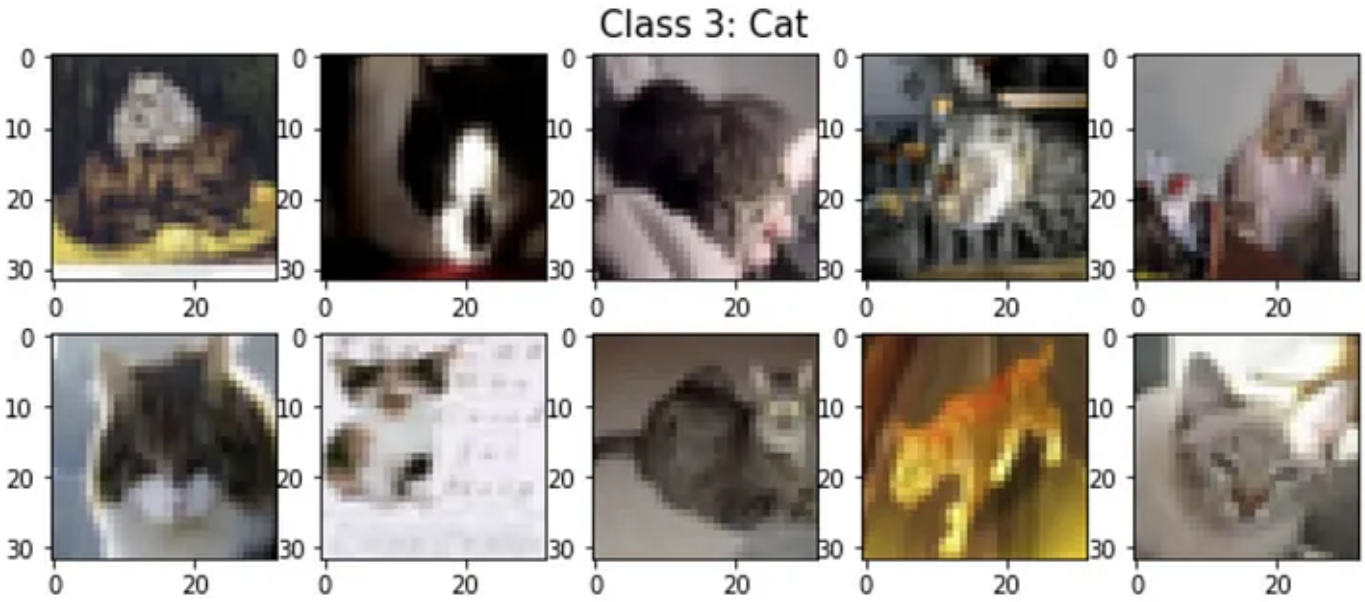
```
array([[   0, 6000],
       [   1, 6000],
       [   2, 6000],
       [   3, 6000],
       [   4, 6000],
       [   5, 6000],
       [   6, 6000],
       [   7, 6000],
       [   8, 6000],
       [   9, 6000]], dtype=int64)
```

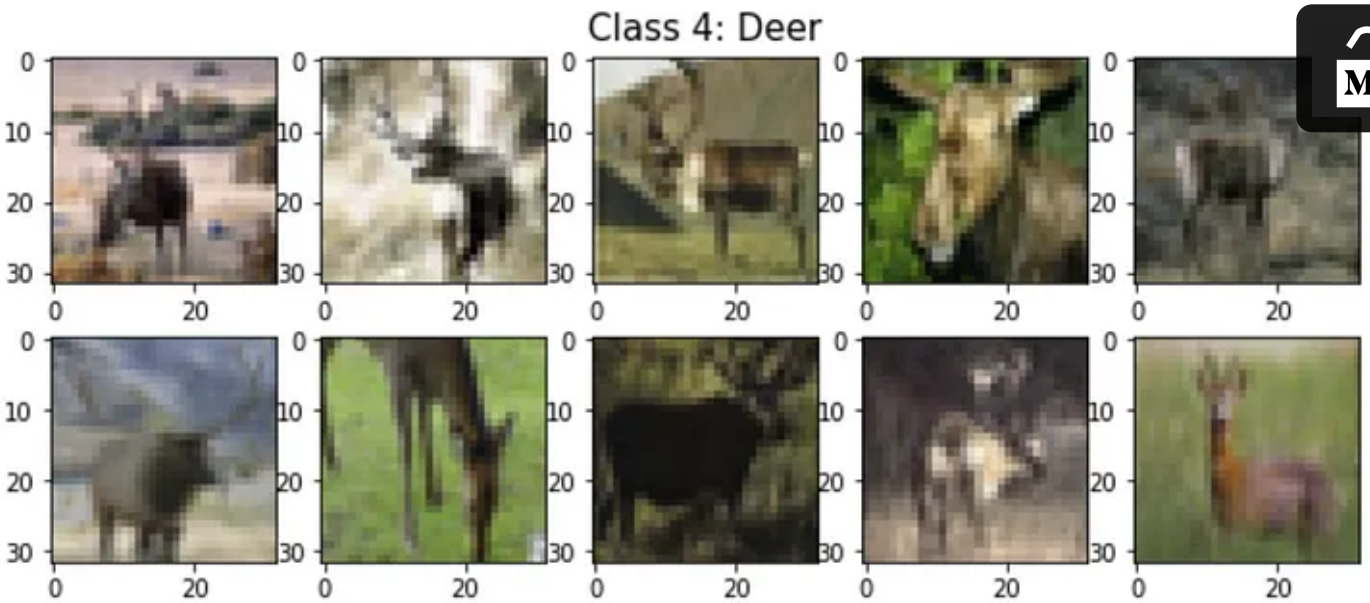
```
# Plot Class N (0-9)

TARGET = # Class index here
NUM_ARRAYS = 10

arrays = X[np.where(y==TARGET)]
random_arrays_indices = np.random.choice(len(arrays),NUM_ARRAYS)
random_arrays = arrays[random_arrays_indices]

fig = plt.figure(figsize=[NUM_ARRAYS,4])
plt.title('Class 0: Plane',fontsize = 15)
plt.axis('off')

for index in range(NUM_ARRAYS):
     fig.add_subplot(2, int(NUM_ARRAYS/2), index+1)
     plt.imshow(random_arrays[index])
```

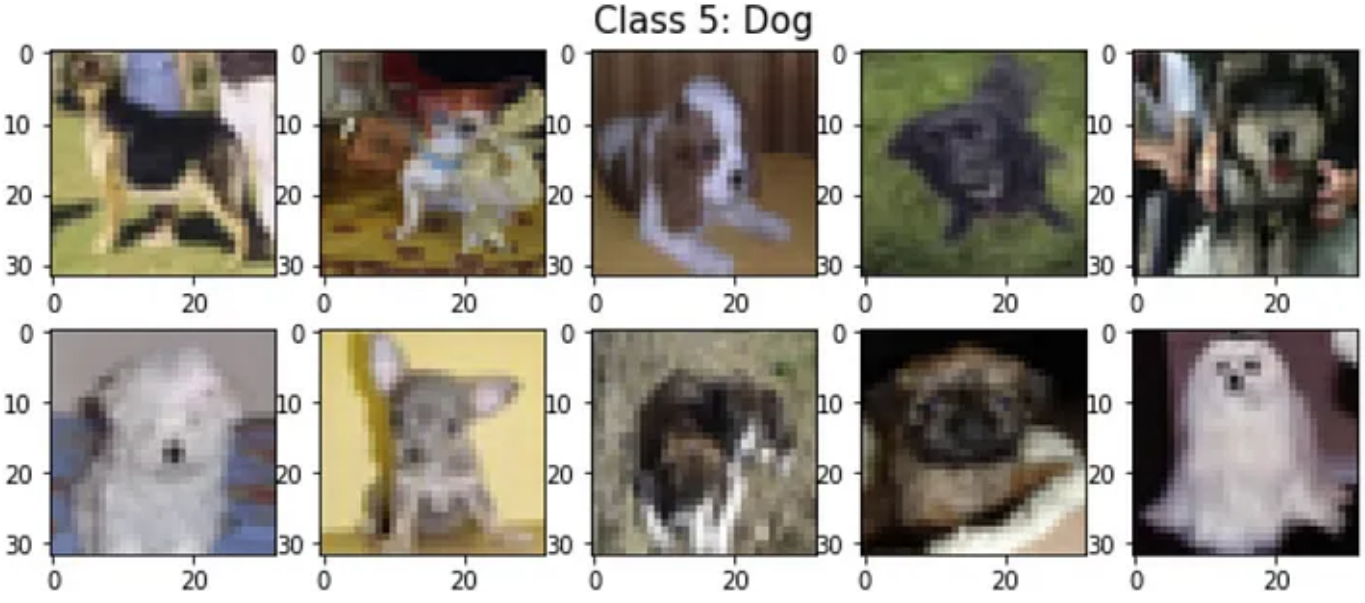


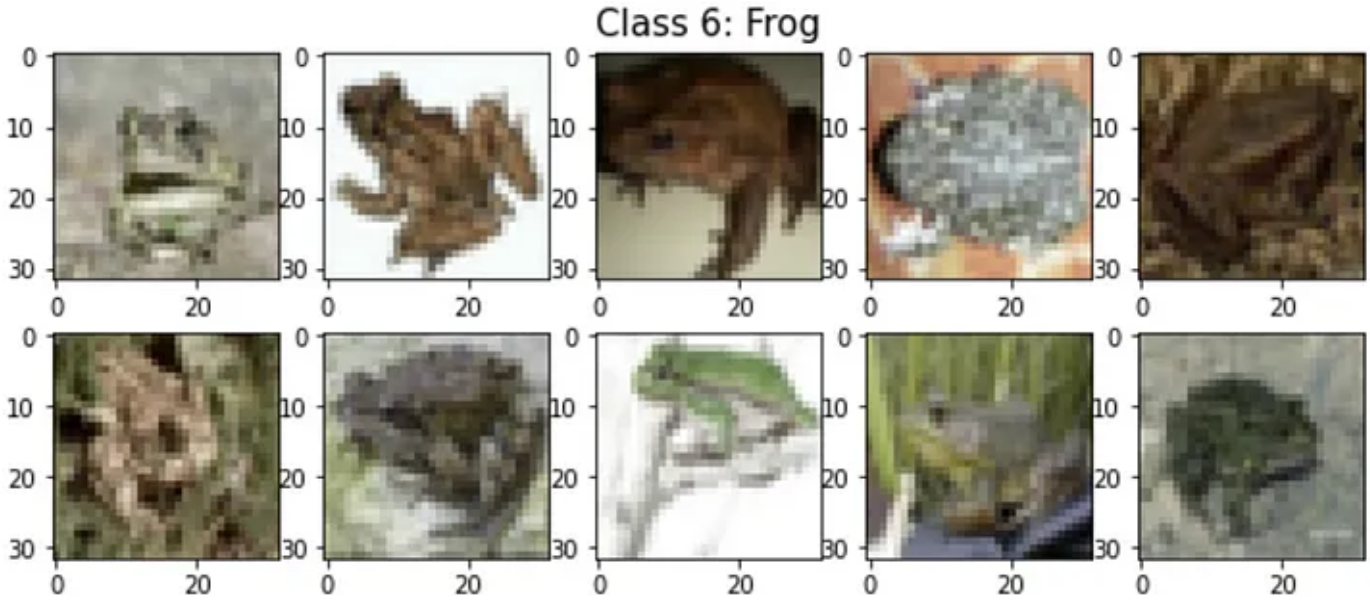

Class 1: Car

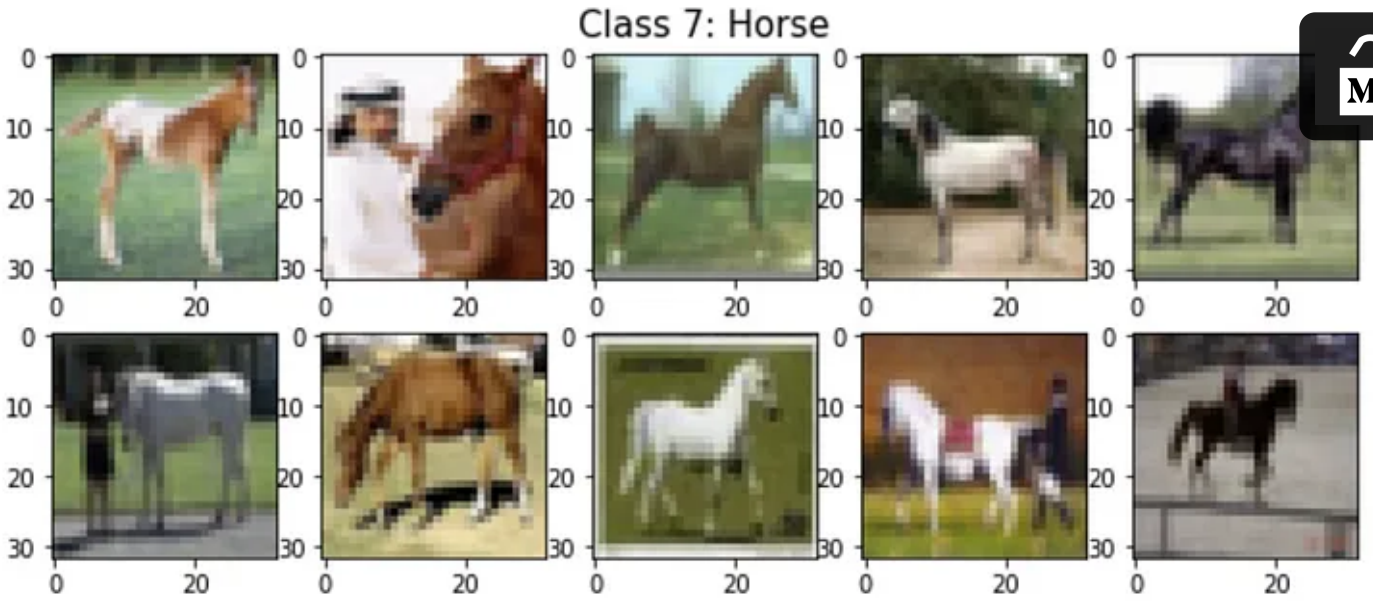

Class 2: Bird

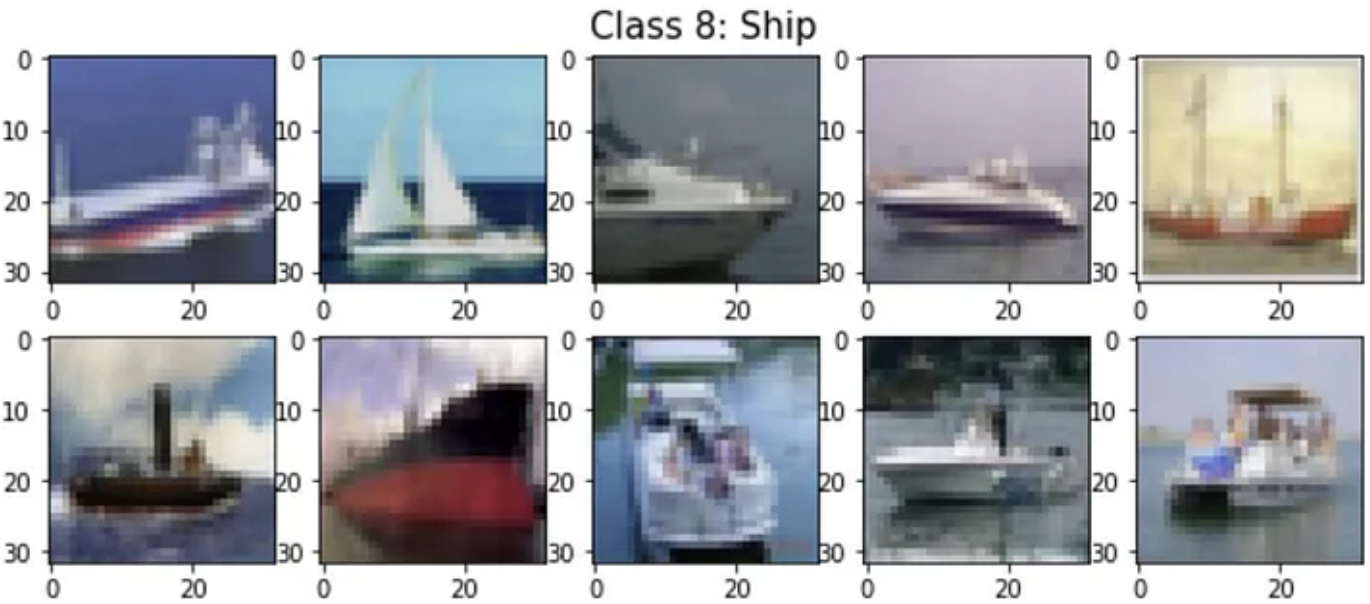

Class 3: Cat

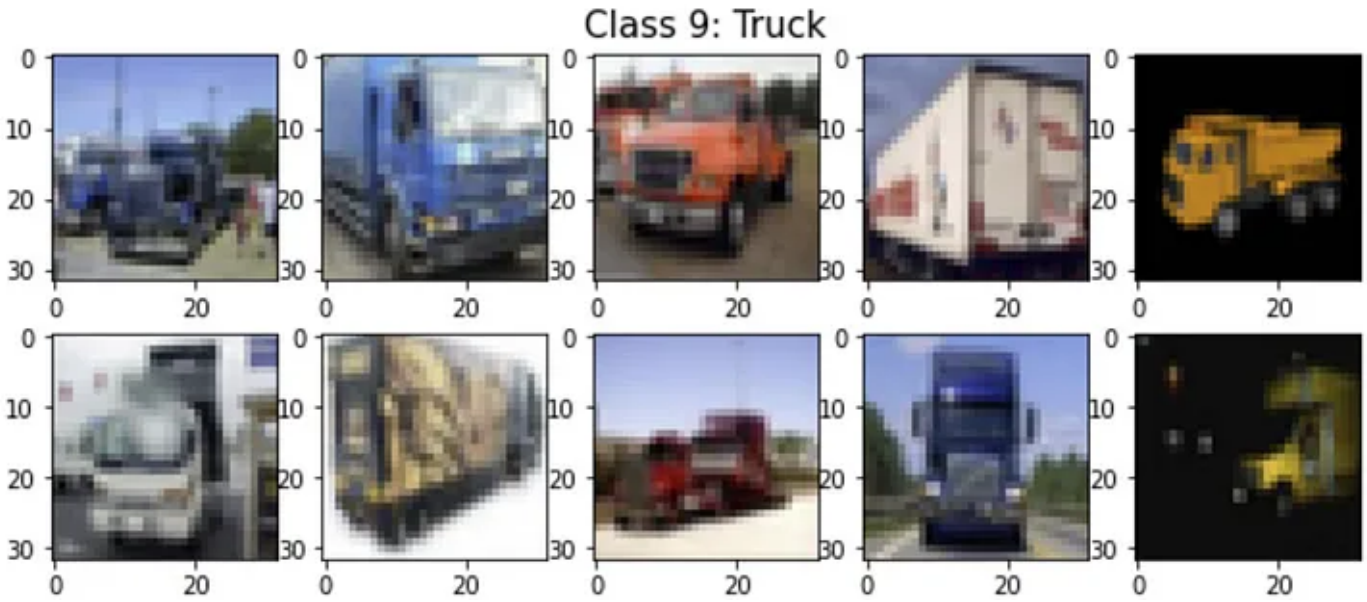

Class 4: Deer


Class 5: Dog


Class 6: Frog


Class 7: Horse


Class 8: Ship



2.2 Generating Triplets

```
# initialize triplets array
triplets = np.empty((0,3,32,32,3),dtype=np.uint8)

# get triplets for each class
for target in range(10):

    locals()['arrays_'+str(target)] = X[np.where(y==target)].reshape(3000,2,32,32
    locals()['arrays_not_'+str(target)] = X[np.where(y!=target)]

    random_indices = np.random.choice(len(locals()['arrays_not_'+str(target)]),30(
    locals()['arrays_not_'+str(target)] = locals()['arrays_not_'+str(target)][ran(

    locals()['arrays_'+str(target)] = np.concatenate(
        [
            locals()['arrays_'+str(target)],
            locals()['arrays_not_'+str(target)].reshape(3000,1,32,32,3)
        ],
        axis = 1
    )

    triplets = np.concatenate([triplets,locals()['arrays_'+str(target)]],axis=0)

# check triplets size
assert triplets.shape == (30000,3,32,32,3)

# plot triplets array to visualize
TEST_SIZE = 5
random_indices = np.random.choice(len(triplets),TEST_SIZE)
```

```
fig = plt.figure(figsize=[5,2*TEST_SIZE])
plt.title('ANCHOR | POSITIVE | NEGATIVE',fontsize = 15)
plt.axis('off')

for row,i in enumerate(range(0,TEST_SIZE*3,3)):
    for j in range(1,4):
        fig.add_subplot(TEST_SIZE, 3, i+j)
        random_index = random_indices[row]
        plt.imshow(triplets[random_index,j-1])

# save triplet array
np.save('triplets_array.npy',triplets)
```

2.3 Preparing for Model Training/Evaluation

```
# Import all libraries

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras import Input, optimizers, Model
from tensorflow.keras.layers import Layer, Lambda
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import backend as K
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import plot_model

from sklearn.metrics import precision_recall_curve, roc_curve, roc_auc_score
from sklearn.model_selection import train_test_split

from scipy import spatial
```

```
triplets = np.load('triplets_array.npy')

triplets = triplets/255 #normalize by 255
labels = np.ones(len(triplets)) #create a fixed label

assert triplets.shape == (30000,3,32,32,3)
```

```
# Split data into our train and test set

X_train, X_test, y_train, y_test = train_test_split(
    triplets,
    labels,
    test_size=0.05,
    random_state=42
)
```

```
# Load pretrained model for transfer learning

pretrained_model = MobileNetV2(
    weights='imagenet',
    include_top=False,
    input_shape=(32,32,3)
)

for layer in pretrained_model.layers:
    layer.trainable = True
```

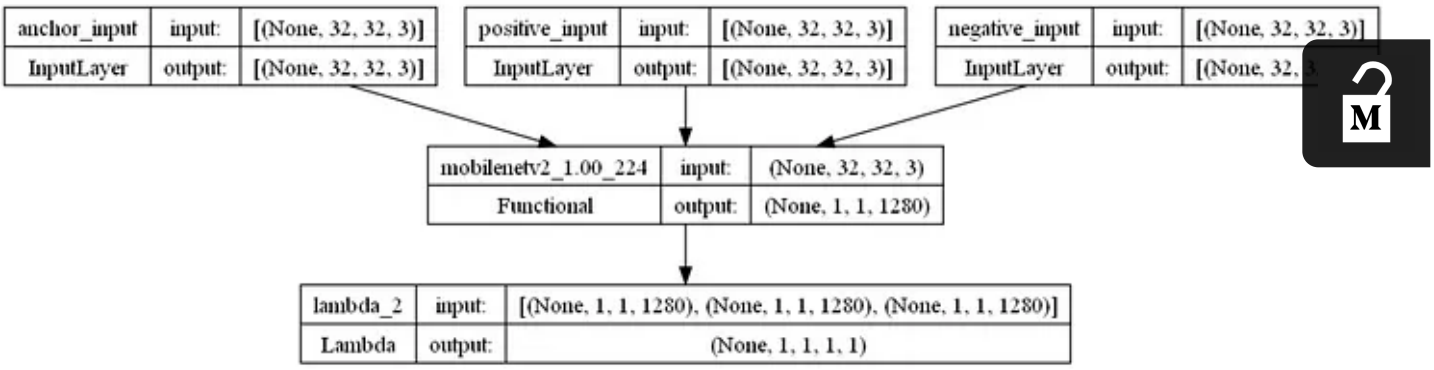
2.4 Model Training

```
# Initialize functions for Lambda Layer

def cosine_distance(x,y):
    x = K.l2_normalize(x, axis=-1)
    y = K.l2_normalize(y, axis=-1)
    distance = 1 - K.batch_dot(x, y, axes=-1)
    return distance

def triplet_loss(templates, margin=0.4):

    anchor,positive,negative = templates

    positive_distance = cosine_distance(anchor,positive)
    negative_distance = cosine_distance(anchor,negative)

    basic_loss = positive_distance-negative_distance+margin
    loss = K.maximum(basic_loss,0.0)

    return loss
```

```
# Adopting the TensorFlow Functional API

anchor = Input(shape=(32, 32,3), name='anchor_input')
A = pretrained_model(anchor)

positive = Input(shape=(32, 32,3), name='positive_input')
P = pretrained_model(positive)

negative = Input(shape=(32, 32,3), name='negative_input')
N = pretrained_model(negative)

loss = Lambda(triplet_loss)([A, P, N])

model = Model(inputs=[anchor,positive,negative],outputs=loss)
```

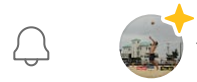
```
# Create a custom loss function since there are no ground truths label

def identity_loss(y_true, y_pred):
    return K.mean(y_pred)
```

```
model.compile(loss=identity_loss, optimizer=Adam(learning_rate=1e-4))

callbacks=[EarlyStopping(
    patience=2,
    verbose=1,
    restore_best_weights=True,
    monitor='val_loss'
    )]

# view model
plot_model(model, show_shapes=True, show_layer_names=True, to_file='siamese_triple
```




```
# Start training - y_train and y_test are dummy

model.fit(
    [X_train[:,0],X_train[:,1],X_train[:,2]],
    y_train,
    epochs=50,
    batch_size=64,
    validation_data=([X_test[:,0],X_test[:,1],X_test[:,2]],y_test),
    callbacks=callbacks
)
```

```
Epoch 1/50
446/446 [==============================] - 24s 41ms/step - loss: 0.3392 - val_loss: 0.3363
Epoch 2/50
446/446 [==============================] - 17s 38ms/step - loss: 0.2293 - val_loss: 0.2565
Epoch 3/50
446/446 [==============================] - 17s 38ms/step - loss: 0.1866 - val_loss: 0.2212
Epoch 4/50
446/446 [==============================] - 17s 38ms/step - loss: 0.1620 - val_loss: 0.1795
Epoch 5/50
446/446 [==============================] - 17s 39ms/step - loss: 0.1408 - val_loss: 0.1680
```

```
Epoch 10/50
446/446 [==============================] - 17s 39ms/step - loss: 0.0779 - val_loss: 0.1318
Epoch 11/50
446/446 [==============================] - 18s 39ms/step - loss: 0.0722 - val_loss: 0.1301
Epoch 12/50
446/446 [==============================] - 18s 41ms/step - loss: 0.0656 - val_loss: 0.1216
Epoch 13/50
446/446 [==============================] - 18s 40ms/step - loss: 0.0599 - val_loss: 0.1257
Epoch 14/50
446/446 [==============================] - 17s 39ms/step - loss: 0.0551 - val_loss: 0.1198
Epoch 15/50
446/446 [==============================] - 17s 39ms/step - loss: 0.0505 - val_loss: 0.1128
Epoch 16/50
446/446 [==============================] - 18s 39ms/step - loss: 0.0463 - val_loss: 0.1175
Epoch 17/50
445/446 [==============================>.] - ETA: 0s - loss: 0.0455Restoring model weights from the end of the best epoch: 15.
446/446 [==============================] - 17s 39ms/step - loss: 0.0455 - val_loss: 0.1187
Epoch 17: early stopping
```

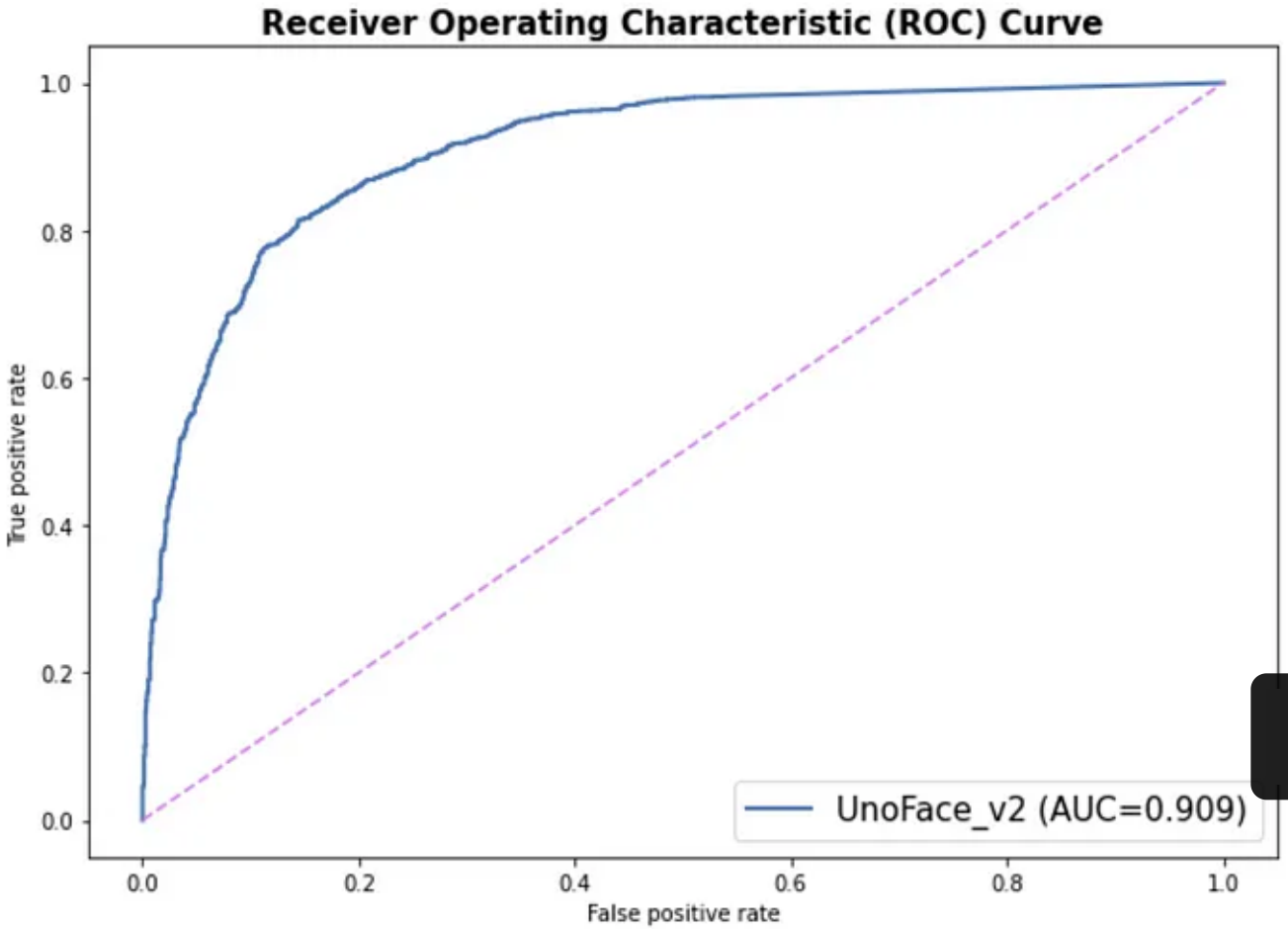
2.5 Model Evaluation

```
X_test_anchor = X_test[:,0]
X_test_positive = X_test[:,1]
X_test_negative = X_test[:,2]

# extract the CNN model for inference
siamese_model = model.layers[3]

X_test_anchor_template = np.squeeze(siamese_model.predict(X_test_anchor))
X_test_positive_template = np.squeeze(siamese_model.predict(X_test_positive))
X_test_negative_template = np.squeeze(siamese_model.predict(X_test_negative))

y_test_targets = np.concatenate([np.ones((len(X_test),)),np.zeros((len(X_test),))]
```

```
# Get predictions in angular similarity scores

def angular_similarity(template1,template2):

    score = np.float32(1-np.arccos(1-spatial.distance.cosine(template1,template2))
```

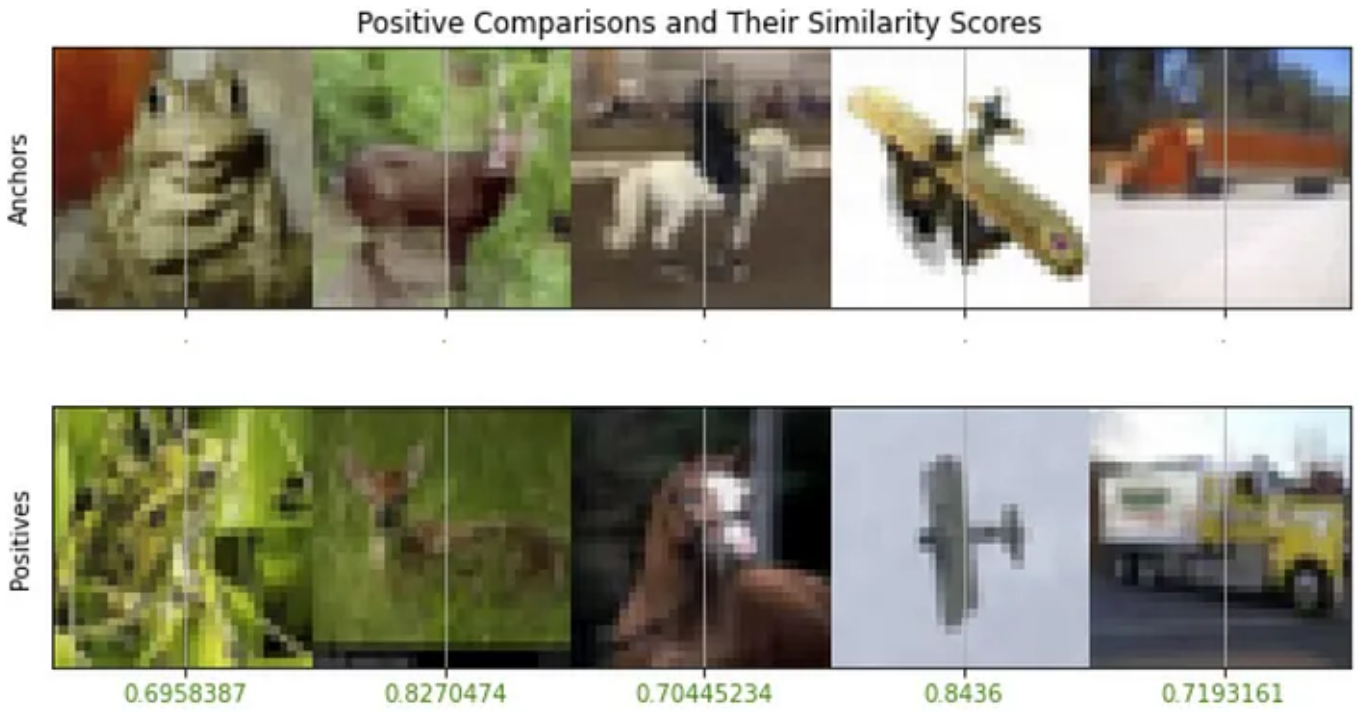
```
    return score

y_predict_targets = []

for index in range(len(X_test)):
    similarity = angular_similarity(X_test_anchor_template[index],X_test_positive
    y_predict_targets.append(similarity)

for index in range(len(X_test)):
    similarity = angular_similarity(X_test_anchor_template[index],X_test_negative
    y_predict_targets.append(similarity)
```

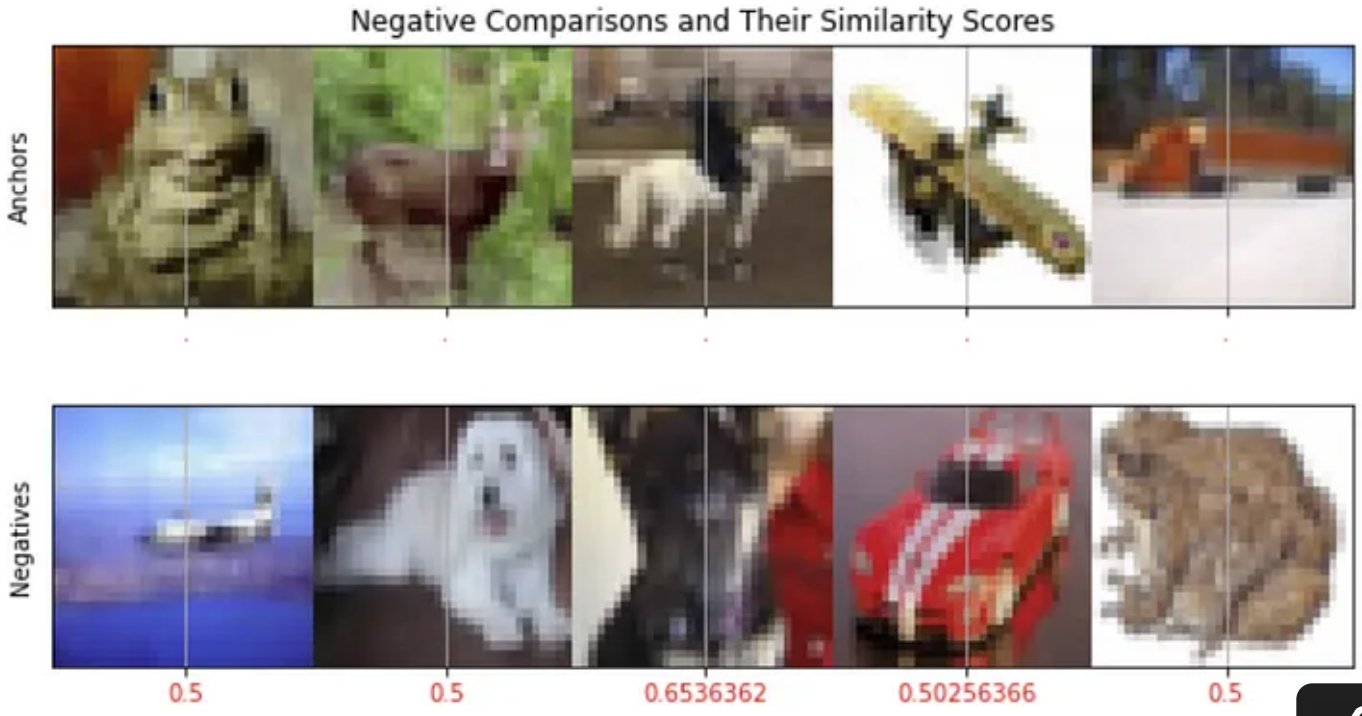
```
# Get prediction results with ROC Curve and AUC scores

fpr, tpr, thresholds = roc_curve(y_test_targets, y_predict_targets)

fig = plt.figure(figsize=[10,7])
plt.plot(fpr, tpr,lw=2,label='UnoFace_v2 (AUC={:.3f})'.format(roc_auc_score(y_tes
plt.plot([0,1],[0,1],c='violet',ls='--')
plt.xlim([-0.05,1.05])
plt.ylim([-0.05,1.05])
plt.legend(loc="lower right",fontsize=15)

plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('Receiver Operating Characteristic (ROC) Curve',weight='bold',fontsize=
```




```
# Getting Test Pairs and their Corresponding Predictions

positive_comparisons = X_test[:,[0,1]]
negative_comparisons = X_test[:,[0,2]]

positive_predict_targets = np.array(y_predict_targets)[:1500]
negative_predict_targets = np.array(y_predict_targets)[1500:]

assert positive_comparisons.shape == (1500,2,32,32,3)
assert negative_comparisons.shape == (1500,2,32,32,3)

assert positive_predict_targets.shape == (1500,)
assert negative_predict_targets.shape == (1500,)

np.random.seed(21)
NUM_EXAMPLES = 5
random_index = np.random.choice(range(len(positive_comparisons)),NUM_EXAMPLES)
```

```
# Plotting Similarity Scores for Positive Comparisons
# (Switch values and input to plot for Negative Comparisons)

plt.figure(figsize=(10,4))
plt.title('Positive Comparisons and Their Similarity Scores')
plt.ylabel('Anchors')
plt.yticks([])
plt.xticks([32*x+16 for x in range(NUM_EXAMPLES)], ['.' for x in range(NUM_EXAMPLI
for i,t in enumerate(plt.gca().xaxis.get_ticklabels()):
    t.set_color('green')
plt.grid(None)
anchor = np.swapaxes(positive_comparisons[:,0][random_index],0,1)
anchor = np.reshape(anchor,[32,NUM_EXAMPLES*32,3])
plt.imshow(anchor)

plt.figure(figsize=(10,4))
plt.ylabel('Positives')
plt.yticks([])
plt.xticks([32*x+16 for x in range(NUM_EXAMPLES)], positive_predict_targets[r
for i,t in enumerate(plt.gca().xaxis.get_ticklabels()):
    t.set_color('green')
plt.grid(None)
positive = np.swapaxes(positive_comparisons[:,1][random_index],0,1)
positive = np.reshape(positive,[32,NUM_EXAMPLES*32,3])
plt.imshow(positive)
```





3. Conclusion

Congratulations on completing the theory and code-along! I hope this tutorial has provided an all-around useful introduction to Siamese Networks and their application to object similarity.

Before we end, I should also add that how the object similarity scores are processed depends on the problem statement.

If we are doing a 1:1 object comparison during production (whether 2 objects are similar or different), then usually a similarity threshold must be set based on the level of False Match Rate (FMR) at test time. On the other hand, if we are doing 1:N object matching, then usually the objects with the highest similarity scores are returned and ranked.

Note: For the complete codes, check out my GitHub.

With that, I thank you for your time and hope you enjoyed this tutorial. I would also like to end off by introducing you to an extremely important topic of data-centric machine learning that is elaborated on in this article:

Data-Centric AI — Data Collection and Augmentation Strategy

A comprehensive guide to data generation strategy for data-centric Machine Learning projects

pub.towardsai.net

> *Thanks for reading! If you have enjoyed the content, pop by my other articles on Medium and follow me on LinkedIn.*
>
> **Support me!** — *If you are* not *subscribed to Medium, and like my content, do consider supporting me by joining Medium via my referral link.*

Join Medium with my referral link - Tan Pengshi Alvin

Read every story from Tan Pengshi Alvin (and thousands of other writers on Medium). Your membership fee directly...

tanpengshi.medium.com

Artificial Intelligence | Neural Networks | Deep Learning | Machine Learning | Data Science

Follow

Written by Tan Pengshi Alvin

456 Followers · Writer for Towards Data Science

Data Scientist, ML and Software Engineer. Shares codes, technical data science concepts and ideas. LinkedIn: https://www.linkedin.com/in/tanpengshi/

More from Tan Pengshi Alvin and Towards Data Science

Tan Pengshi Alvin in Towards Data Science

Introduction to Text Summarization with ROUGE Scores

An informative guide to summarization in Natural Language Processing (NLP)

· 8 min read · Mar 21, 2022

Bex T. in Towards Data Science

130 ML Tricks And Resources Curated Carefully From 3 Years (Plus Free eBook)

Each one is worth your time

· 48 min read · Aug 1

Kenneth Leung in Towards Data Science

Running Llama 2 on CPU Inference Locally for Document Q&A

Clearly explained guide for running quantized open-source LLM applications on CPUs using LLama 2, C Transformers, GGML, and LangChain

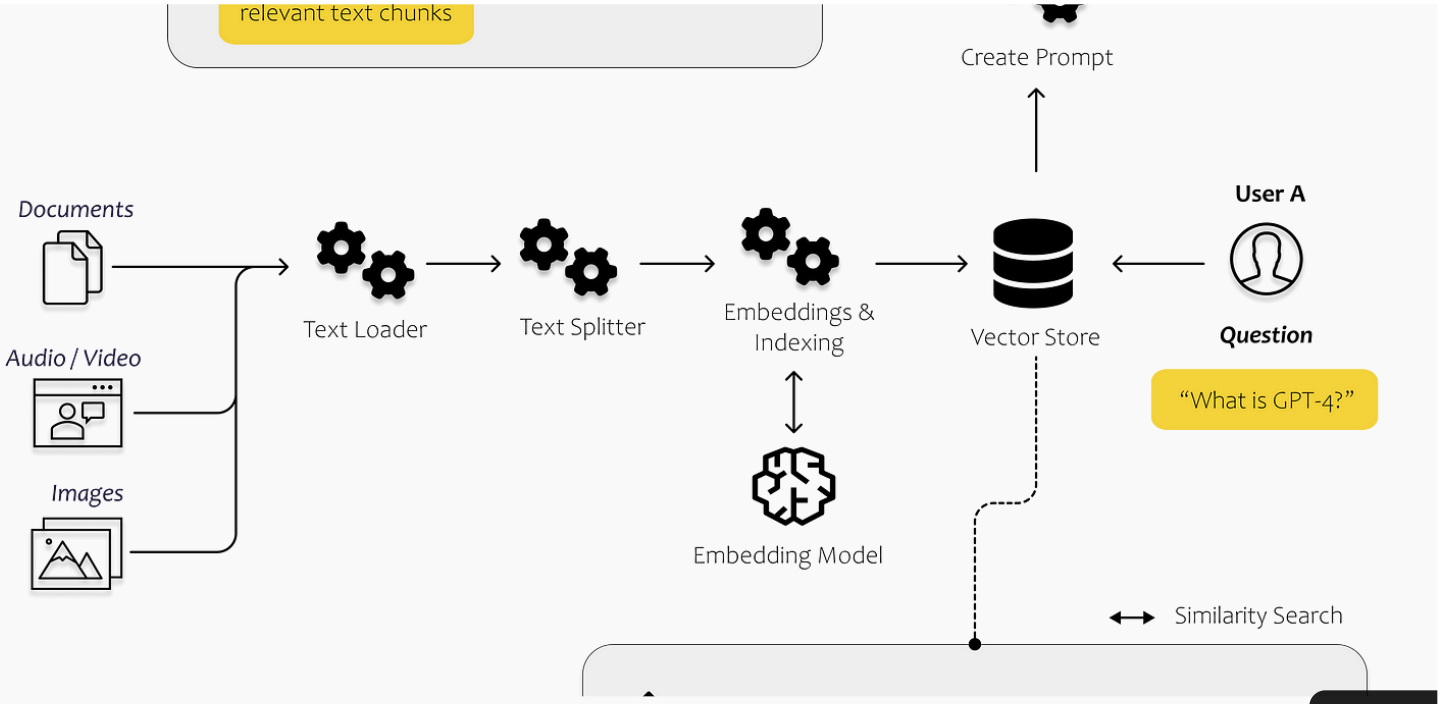
· 11 min read · Jul 18

Tan Pengshi Alvin in Towards Data Science

Serving TensorRT Models with NVIDIA Triton Inference Server

Achieving optimal throughput and latency with model inference on high client-server traffic

· 6 min read · Dec 15, 2022



Dominik Polzer in Towards Data Science

All You Need to Know to Build Your First LLM App

A step-by-step tutorial to document loaders, embeddings, vector stores and prompt templates

· 26 min read · Jun 22

4.3K 40

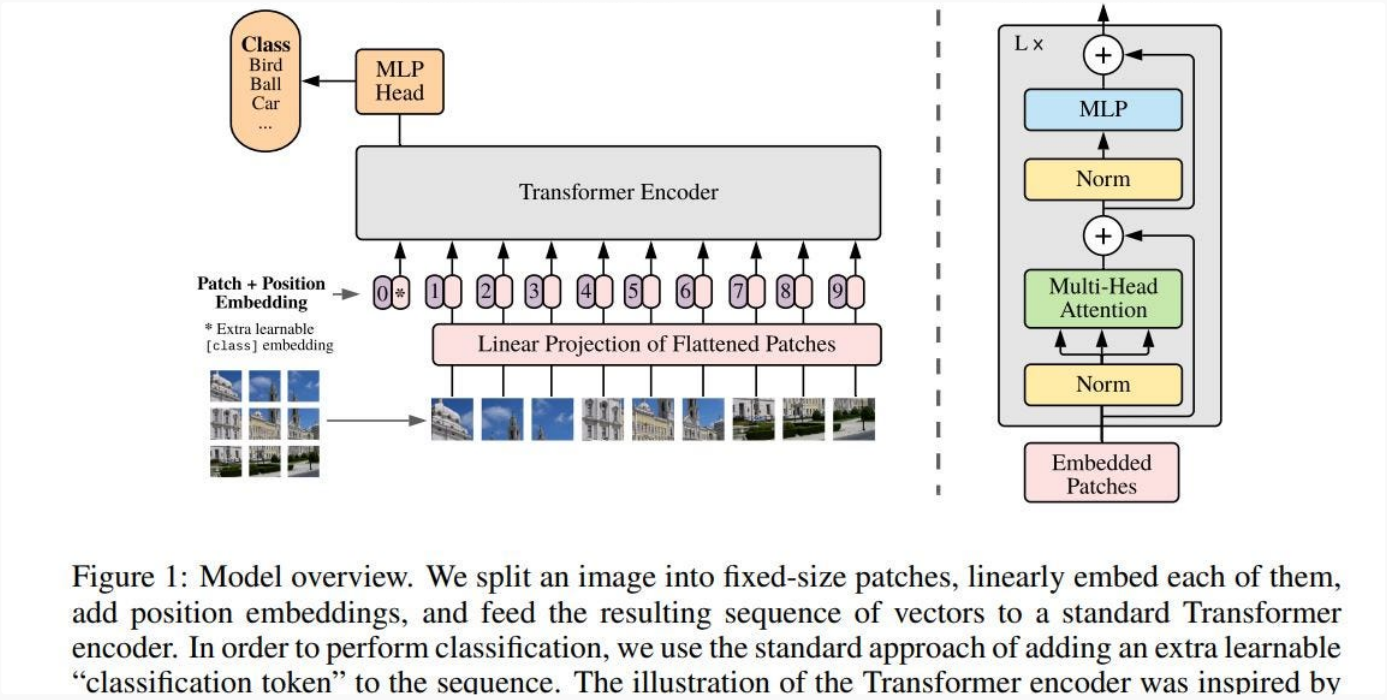


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by

Fahim Rustamy, PhD

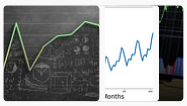
Vision Transformers vs. Convolutional Neural Networks

This blog post is inspired by the paper titled AN IMAGE IS WORTH 16×16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE from google's...

7 min read · Jun 4

220 4

Lists



Predictive Modeling w/ Python

18 stories · 261 saves

Natural Language Processing

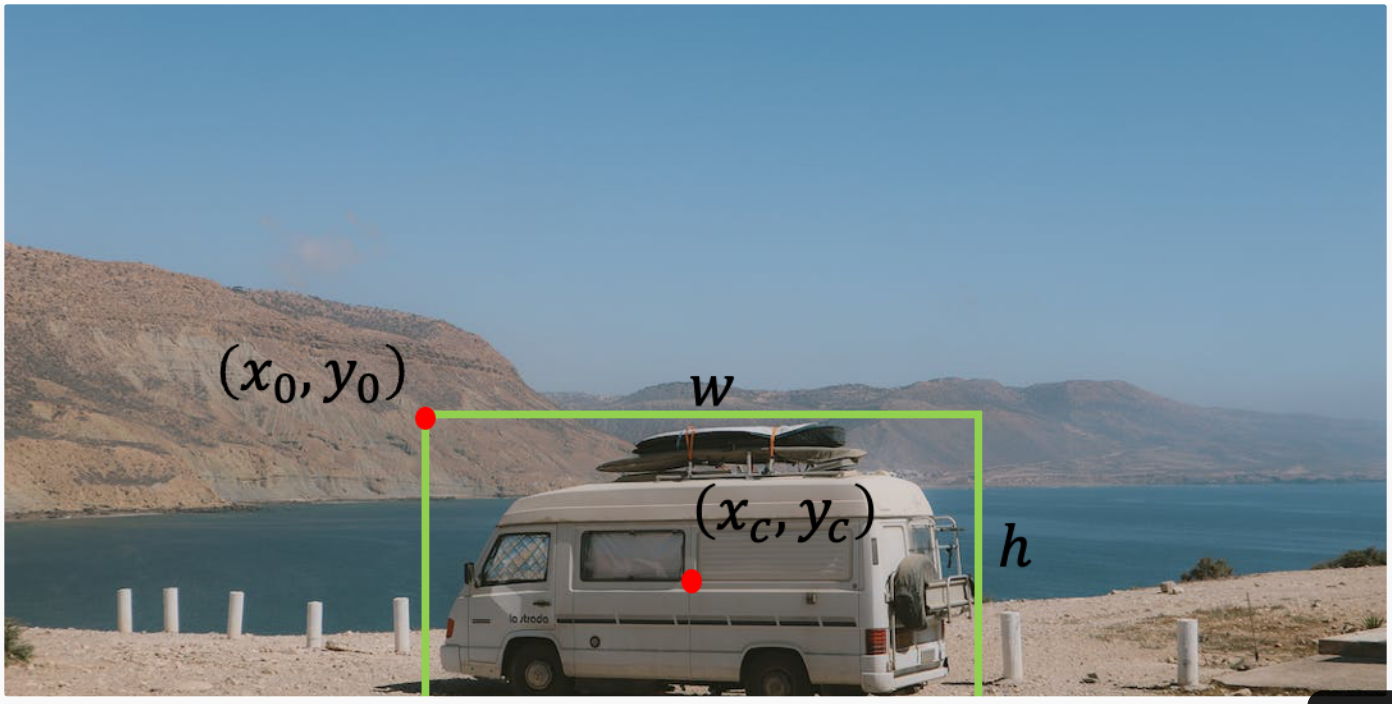
503 stories · 131 saves

Practical Guides to Machine Learning

10 stories · 274 saves

ChatGPT prompts

24 stories · 248 saves



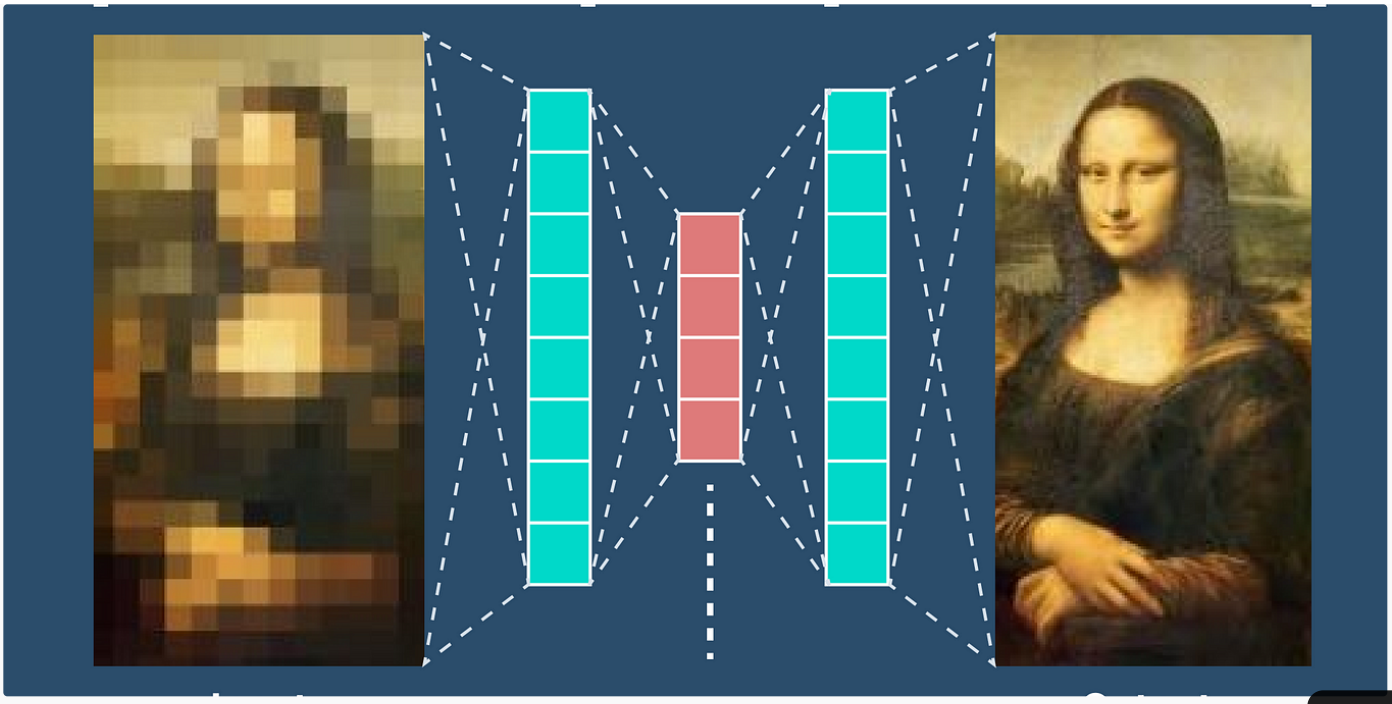
DZ in Python in Plain English

Single Object Detection with PyTorch Step-by-Step

In the task of object detection, we want to find the location of an object in an image. We may search for one type of object (single-object...

14 min read · Aug 5

17



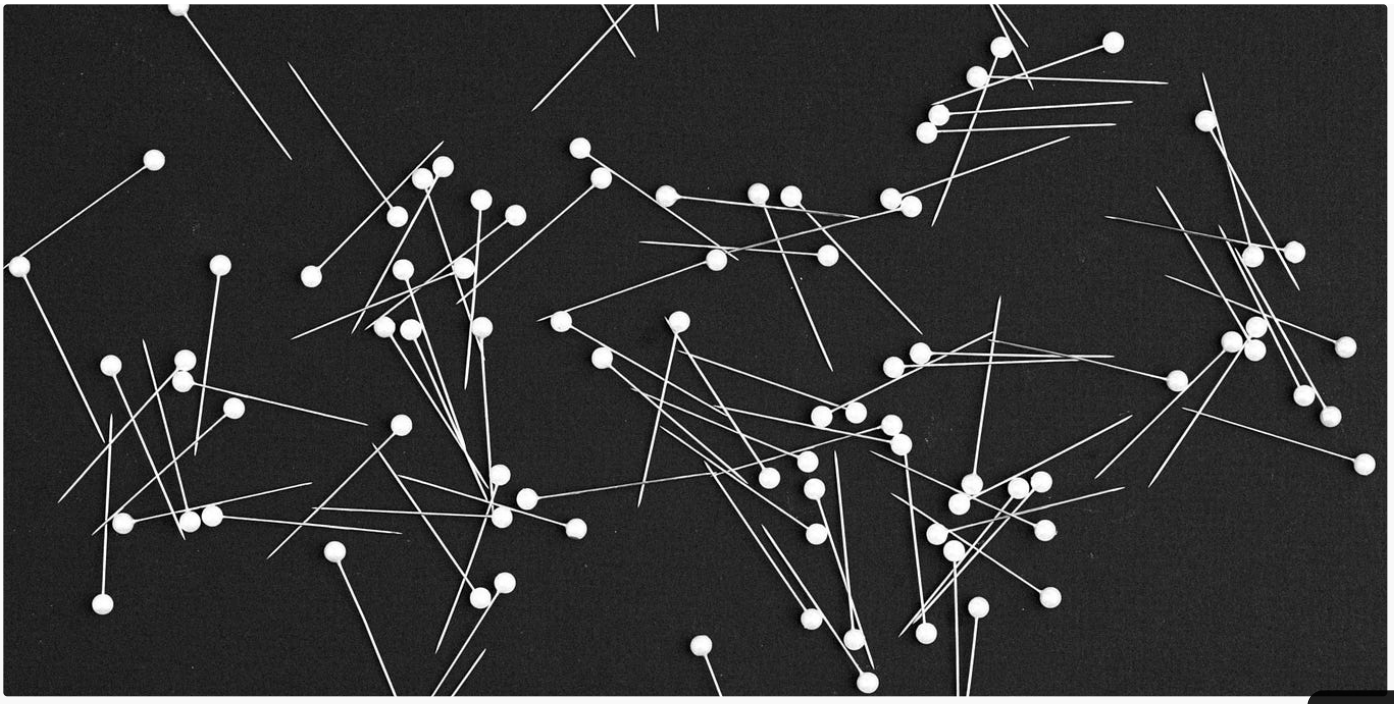
Visheshtaposthali

Autoencoders: Unveiling the Mystery of Data Compression and Reconstruction

Introduction

3 min read · Aug 1

3

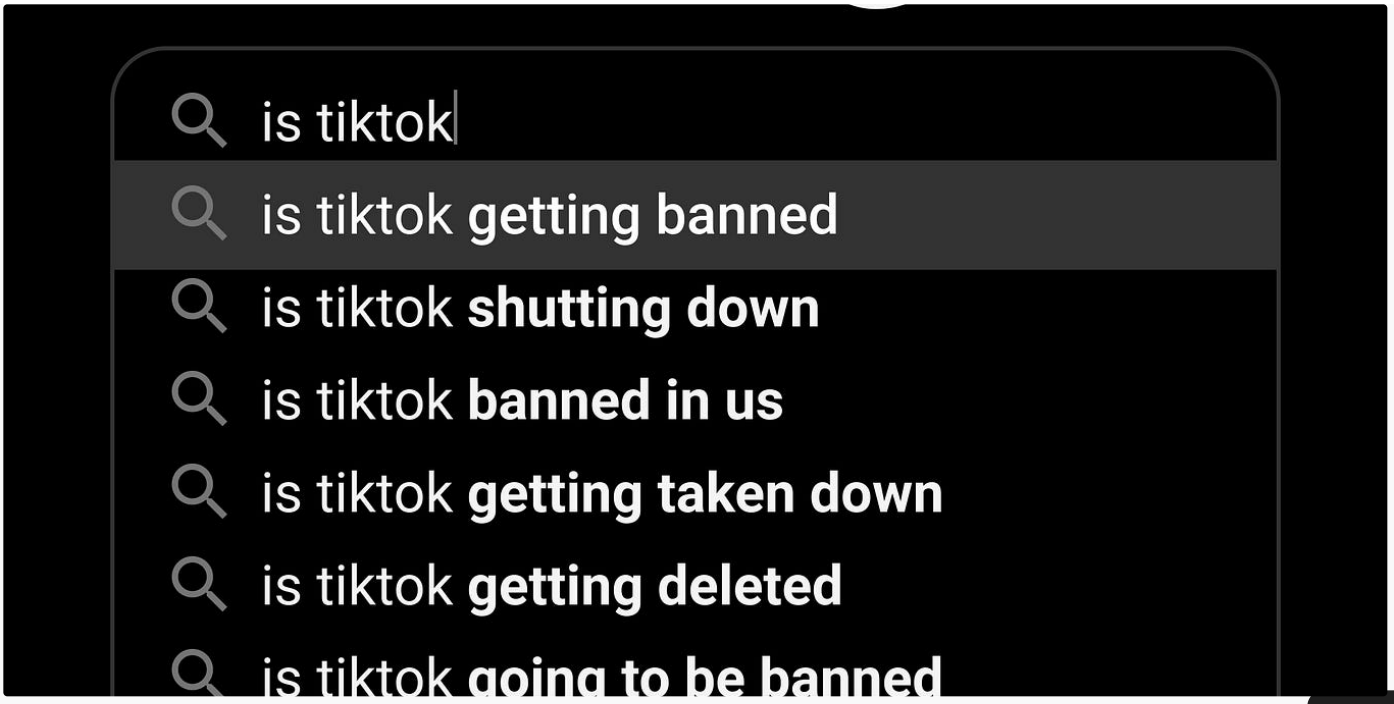


Vikas Kumar Ojha in Geek Culture

Binary Neural Networks: A Game Changer in Machine Learning

This blog explains about binary neural networks which have potential of revolutionizing deep learning if proper efforts are made.

· 9 min read · Feb 19



Christian Bernecker

NLP SIMILARITY: Use pretrained word embeddings for semantic similarity search with BERT

Use pretrained word embeddings to measure document similarity and do semantic similarity search with a BERT Transformer.

5 min read · Mar 1

67 2

See more recommendations