UNIVERSITY OF CAMBRIDGE

**MPhil in Machine Learning and Machine Intelligence 2022-2023**

# Module Coursework Feedback

Module Title: Computer Vision

Module Code: MLMI12

Candidate Number: K5038

Coursework Number: Mini Project

***I confirm that this piece of work is my own unaided effort and adheres to the Department of Engineering's guidelines on plagiarism.*** √

Date Marked: Marker's Name(s):

**Marker's Comments:**

**This piece of work has been completed to the following standard** *(Please circle as appropriate)*:

| | **Distinction** | | | **Pass** | | | **Fail (C+ - marginal fail)** | | |
|---|---|---|---|---|---|---|---|---|---|
| **Overall assessment (circle grade)** | **Outstanding** | **A+** | **A** | **A-** | **B+** | **B** | **C+** | **C** | **Unsatisfactory** |
| **Guideline mark (%)** | 90-100 | 80-89 | 75-79 | 70-74 | 65-69 | 60-64 | **55-59** | **50-54** | **0-49** |
| **Penalties** | **10% of mark for each day, or part day, late (Sunday excluded).** | | | | | | | | |

The assignment grades are given **for information only**; results are provisional and are subject to confirmation at the Final Examiners Meeting and by the Department of Engineering Degree Committee.

# MLMI12 Mini Project:
# Image Matching using Deep Learning

K5038

## 1. Introduction

Image Matching is the process of determining whether or not two images are similar. Similarity can be based upon a number of variables. It could be the subject of the images, objects present within the image, location, colour, or a mixture of multiple attributes. In general, similarity is quantified based on the specific task or purpose, but in a broad sense, it is regarded as how a human observer would compare two photographs. In a computational sense it might seem that we could simply compare each pixel of two images to determine their similarity, however this simple approach is highly prone to image transformations, and visual subjectivity. Therefore, it is necessary to design effective image representations for similarity matching.

Prior to deep learning techniques, image matching was performed using image descriptors [2]. A descriptor is a collection of attributes that can uniquely define all of an image's characteristics. Descriptors must be created so that they are invariant to simple augmentations, i.e., they should remain the same even if the image is flipped or rotated to a given degree. Deriving such unique manually constructed descriptors is an extremely difficult task. Deep learning methods make this task of summarising images much simpler due to their inherent nature of embedding and mapping key attributes. As an image is processed through the layers of a neural network, each layer learns unique attributes, such as edges, brightness features, or object categories. Essentially, the network compresses the image and generates a single vector of numbers that can represent all of the image's vital information. This vector is basically the learned descriptor for the particular image. A neural network embeds an image onto an n-dimensional space, and the vector is the image's coordinate in this space. So, more similar images are mapped to the same location. To determine whether or not two images are similar, we can simply calculate the distance between their vectors.

In this project we design and train a neural network for the task of image matching on the Tiny ImageNet dataset [1]. There are various approaches for image similarity learning in deep neural networks. Similarity can be learned through a two step distance metric, using an end-to-end siamese network, or through end-to-end feature concatenation. We train and evaluate some of these methods on the provided supervised dataset. Our inference objective is given two images, perform a binary classification of whether the images are similar or dissimilar.

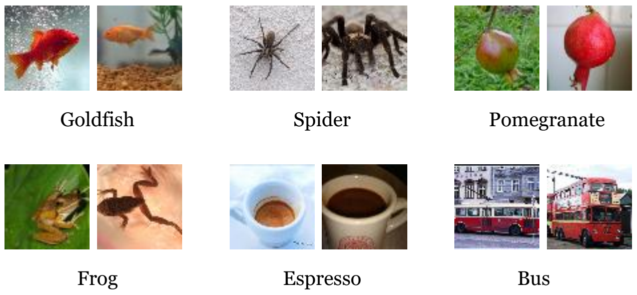

Figure 1: Sample images of different classes from the Tiny ImageNet dataset.

## 2. Dataset Preparation

The Tiny ImageNet [1] dataset is a smaller version of the larger LSVRC ImageNet 1K dataset [4]. Some samples of the classes are shown in Fig. 1. We can see that even among the same classes, there are significant differences in the images' shape, colour, background, and orientation. The dataset consists of 200 classes with 500 train and 50 validation images per class. There is also a separate, unlabeled test set, but we will not use it for our image matching task. Due to limited resources, instead of using the whole dataset, we randomly selected 50 classes to expedite our experiments. We use all available train images for each class. From the original validation set we separate 20 images per class for testing and the remaining 30 for validation. In addition, we evaluate and analyse model performance on both seen (images of this category were included in the training set) and unseen (images of this category were not included in the training set) categories. For the unseen labels, we randomly selected 10 classes from the primary dataset's 100 unused classes. Using random permutations of the seen and unseen test classes, we generate pairs of similar and dissimilar images to construct the final test set. While training data is left as single images, testing is conducted on image pairings labelled as 1 for similar and 0 for dissimilar. Consequently, the final test of similarity is a binary classification problem. During data partitioning, it was assured that no data leaked between the train, validation, and test sets. The number of samples in each set is summarized in Table 1.

| | | #Class | #Images per class | #Total |
|---|---|---|---|---|
| Train | | 50 | 500 | 25,000 |
| Val | | 50 | 30 | 1500 |
| Test | Seen-Seen | 50-50 | 20 | 1000 |
| | Seen-Unseen | 10-10 | 20 | 200 |
| | Unseen-Unseen | 10-10 | 20 | 200 |

Table 1: Number of samples in the train, validation, and test splits. Here *seen* means images of this category were present in the training set. None of the test images were used for training.

## 3. Architectures

In order to measure the similarity between two images we tested a number of different network architectures and frameworks to evaluate which one performs the best.

### 3.1. Multiclass Classifier Baseline

Since we are working with the Tiny Imagenet dataset, we can make use of the specific characteristics of the data to perform a similarity classification. Most of the images in the dataset contain either only a single object, or in case of multiple objects a single dominant one as shown in Fig. 1. This means we can assign a particular class to the entire image and classify two images as similar if they have both have the same class. This is a baseline heuristic to evaluate which type of networks performs well in capturing the latent space of the data. However, this method will not work if there are multiple dominant classes in the same image. For example, an image containing both a cat and a dog, since we can only assign a single class, one of the images could get the label *cat* and the other one might get *dog*. Thus, even though both images are similar, the different classes will lead to different labels and be identified as dissimilar. The architecture is visualized in Fig. 2.

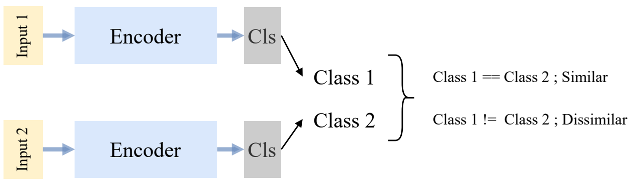

Figure 2: Architecture of the multiclass classifier model for image matching. Here *Cls* means classifier block.

### 3.2. Multiclass Features with Cosine Similarity

This method uses the same multiclass classifier network as the previous one, but instead of using the final classifier labels as determined by the dense layer we utilize the embedded features. The main drawback of using the final classification probabilities is that we can only assign a single label to the image. Continuing with the example of *cat* and *dog*, the classifier might predict the probabilities for both as 50%. But since we can only assign a single label, we select only one randomly. However, the embedded image features that the dense layer is classifying contains equal amount of information for both classes. This embedding feature is a latent space representation of the input image. So, when comparing two images, instead of using the predicted final class probabilities we can compare the feature embeddings directly. We retrieve the feature vectors prior to the final dense layer and calculate their distance using a cosine similarity function. Cosine similarity calculates an euclidean distance between two multidimensional vectors. It produces values between -1 to 1, where 1 means highly similar and -1 means completely dissimilar. We used a threshold of 0.1 to predict the final class. The networks are trained similar to a classification problem like before with images and their corresponding class labels. During inference, we extract the two feature vectors and compute the cosine similarity to perform the final classification. The architecture is shown in Fig. 3

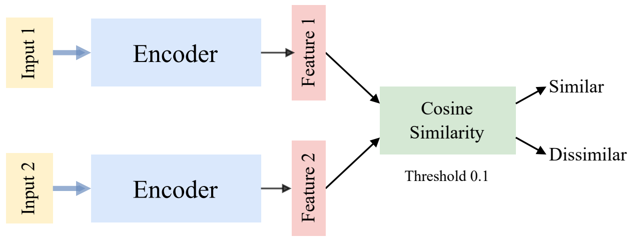

Figure 3: Architecture of the image matching model using multi-class features and cosine similarity.

### 3.3. Siamese Network with Triplet Loss

A siamese neural network is a type of multi network framework that uses shared weights to calculate the feature embeddings of different inputs simultaneously and computes their comparable output vectors. The objective of a siamese learning system is to reduce distance between similar feature vectors and maximize the distance of dissimilar ones. The siamese network is an extension of our previous method where instead of calculating the distance of the feature vectors during inference using cosine similarity, we design the network and the objective function to learn these distances during training. Previously we trained the network on class labels and measured the feature distance during inference. However, we can learn these distances directly during training using a Triplet Loss [5]. During training, we select three images per iteration; and anchor image, a positive image, and a negative image. The anchor is a randomly sampled image from the train set. The positive image is a different image that has the same class label as the anchor, and the negative one is an image with a dif-

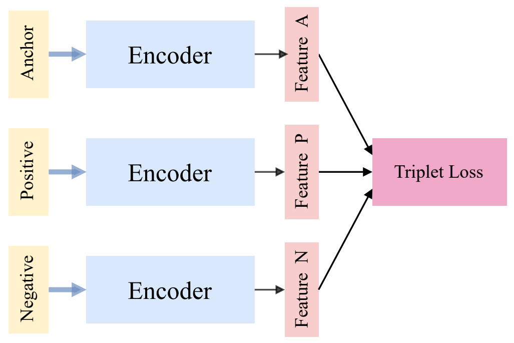

Figure 4: Training of similarity features using Triplet Loss.

ferent class label. We use the same network to calculate the feature vectors for each of these images. We use the Triplet loss over these three feature vectors in order to train the network such that the distance between the anchor and positive image is minimized and the distance between anchor and negative image is maximized. The framework is visualized in Fig. 4. The equation of Triplet loss is as follows;

$$\mathcal{L}_{\text{triplet}} = \left[\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha\right] \tag{1}$$

Here $\|f(A) - f(P)\|^2$ is the distance between the anchor and positive image, and $\|f(A) - f(N)\|^2$ is the distance between anchor and negative image. We are trying to minimize the positive distance and maximize the negative one. $\alpha$ is the margin between the positive and negative pairs. The triplet loss works to tune the network in such a way that the network learns to project the feature vectors to an n-dimensional space where similar features are placed closer together. During inference we follow the same procedure as in Sec. 3.2 by measuring the cosine similarity of the two images and perform thresholding to get the final label.

### 3.4. Merged Binary Classifier

In addition to the above methods, we designed a custom neural network that can directly classify whether two images are similar or not through an end-to-end pipeline. This is simply a combination of the siamese and the classifier approach. The layout of our proposed architecture is shown in Fig. 5. During training we randomly select a pair of anchor-positive or anchor-negative images. Anchor-positive images have a class label 1 meaning they are similar, and anchor-negative have label 0 meaning dissimilar. Both images are sent through the same network to get a pair of feature vectors. Next, these feature vectors are concatenated and passed through a multi-layer dense classifier with a final sigmoid layer that produces a single probability value. Thus, we get a single probability output from our pair of images. The network is trained end-to-end using a Binary Crossentropy loss.The classification head basically learns the differing characteristics between the concatenated feature vectors. This is a much easier task compared to mapping features into a latent space based on similarity. This architecture combines the simplicity of a straightforward classifier with the distance learning of a siamese approach.

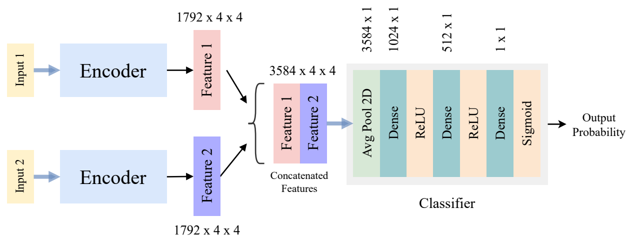

Figure 5: Architecture of the concatenated binary classifier model. The number above the layers represent the size of the feature vector output from that layer.

## 4. Experiments

### 4.1. Training Setup

Although each architecture was trained slightly differently, they all followed the same setup and similar hyper-parameters. The dataset originally consisted of $64 \times 64$ RGB images. Before forwarding to the network, they were resized to $128 \times 128$ and normalized using per channel mean of $(0.485, 0.456, 0.406)$ and std. deviation $(0.229, 0.224, 0.225)$. Moreover, we used additional data augmentation of random horizontal flip, and random rotation of 15 degrees to improve data variation and reduce overfitting. For the binary classifier network, we also used Label Smoothing [3] with a value of 0.01. To train the networks we used Adam optimizer with a learning rate 0.0001 and batch-size 64. We also used Early Stopping on validation loss with a patience of 7 epochs. For all encoders, we used models with pre-trained weights on the Imagenet-1k dataset from the `timm` [6] library. This is applicable because, although Tiny Imagenet is a subset of the Imagenet-1K dataset, they maintain the train and validation splits of the original dataset, i.e Tiny Imagenet validation is made of images from the Imagenet-1K validation set. So models pretrained on the Imagenet-1k that adhere to the data splits can avoid the data leakage. This allowed us to utilize these pretrained models for transfer learning on the similarity task. Lastly, for our language framework we used `PyTorch`, and all our experimentation was performed on the Google Colab platform on Nvidia T4 GPUs.

### 4.2. Encoder Selection

The encoder is responsible for producing the latent space feature vectors from the input images. All our evaluation methods perform similarity matching on top of these feature embeddings. Therefore, it is crucial to have an encoder that

| Model | Test Accuracy | # Params (M) |
|---|---|---|
| VGG-16 | 73.35 | 138.37 |
| ResNet-34 | 77.11 | 21.82 |
| ResNeXT-101 | 83.15 | 83.46 |
| EfficientNet-B0 | 78.66 | 5.29 |
| EfficientNet-B4 | 85.16 | 19.34 |
| ViT-L | 88.26 | 304.53 |
| EfficientNet-L2 | 88.50 | 480.31 |

Table 2: Test accuracy of various encoders on the Tiny Imagenet validation set for multiclass classification. The number of parameters is reported in Millions (M).

can extract the essential image information. To evaluate the performance of various encoders, we examine their multiclass classification performance on the dataset (not similarity matching). If an encoder does well on the baseline; that means it generates features that are more representational of the image classes. The classification results for various encoders are shown in Table 2. We can see that EfficientNet-L2 has the highest performance, however it is also a much larger network with 480 million parameters. This is unfeasible with our training setup. We select EfficientNet-B4 as our preferred encoder due to its comparable performance and reduced parameters, which are required for faster experimentation.

## 4.3. Hyper-parameter Tuning

We evaluated four frameworks in this project for image similarity matching. However, all of these pipelines use the same EfficientNet-B4 (EffB4) encoder as their backbone. So, all hyper-parameter tuning is geared towards optimizing the backbone network. During our experiments, we found the following hyper-parameters to have a significant impact on the final result;

- **Input Size:** The size of the input image that was fed into the network had a considerable impact on the performance. The dataset is originally made of $64 \times 64$ images. For the baseline image matching test with the original images, the network had a test accuracy of $71\%$ on the seen-seen image pairs. The same model when trained on $128 \times 128$ images had an accuracy of $84.25\%$. This is possibly due to the scale and pre-trained nature of the model. EffB4 is a quite large model which was loaded with pre-trained weights that was initially trained on $224 \times 224$ size input. Thus using such low resolution inputs reduces the overall effectiveness of the transfer learning mechanism. Moreover, with such small scale images the model gets overfitted very quickly due to its larger capacity. We settled on using 128 sized images because upsampling to 224 would generate a lot of noise in the data.

- **Embedding Size:** The EffB4 network outputs feature vectors of size $1792 \times 4 \times 4$. These raw features are unsuitable for cosine similarity or triplet loss because these methods need flattened and normalized vectors for distance calculation. So, first we apply $1 \times 1$ average pooling to flatten the raw features and then pass them through a projection module consisting of two dense layers, ReLU activation and a Normalization layer to convert them into $256$ sized feature vectors. Validation losses for various embedding sizes for the triplet loss network is listed in Table 3. Although $256$ sized embeddings have the least validation loss, this is not enough to justify their selection. Theoretically larger embedding size can capture more features of the input, but this also adds more complexity for the objective function as the network needs to optimize the sample feature distances. We perform further test evaluations using both $256$ and $128$ sized embeddings shown in Table 4.

| Embedding Size | Min. Validation Loss |
|---|---|
| 1792 | 0.4912 |
| 512 | 0.2736 |
| 256 | 0.0814 |
| 128 | 0.1943 |

Table 3: Minimum validation loss for siamese training with triplet loss using the EffB4 encoder using various embedding sizes.

- **Learning Rate:** Learning rate did not have a drastic impact on the performance. Since we are transfer learning from pre-trained weights, we used a low rate of 0.0001. With an LR of 0.00001 it took a longer time to reach convergence but resulted in similar test performance. Alternatively an LR of 0.001 resulted in worse performance but relatively faster training. Since we used Adam optimizer which adapts and changes the learning rate as the training progresses, the initial rate had fairly low impact on the final performance.

## 5. Results and Observation

The results of our evaluation of the different models on the image matching test sets are reported in Table 4.

The baseline classification approach using predicted labels is a good method for our dataset and achieves a reasonably high performance on the seen-seen and seen-unseen sets. However, this method completely fails for unseen-unseen classes. Since the network has not seen any of the classes during training, it arbitrarily assigns some random class which results in a mere $62\%$ accuracy. Moreover, since the model has not seen the real classes during training, it can only assign arbitrary ones.

The consequent feature with cosine similarity method is a much better approximation. Since the model is not bound by class labels, the cosine similarity function can better approximate the relative similarities of the feature vectors.

|  | Seen-Seen | Seen-Unseen | Unseen-Unseen |
|---|---|---|---|
| Multiclass Classifier Baseline | 84.25 | 96.28 | 62.00 |
| Multiclass Features + Cosine Sim | 88.70 | 95.11 | 74.38 |
| Siamese + Triplet Loss (256) | 80.14 | 71.06 | **77.12** |
| Siamese + Triplet Loss (128) | 82.36 | 78.12 | 75.00 |
| End-to-End Binary Classifier | **91.03** | **95.09** | 75.04 |

Table 4: Test accuracy of the image matching task on three different image pair sets for the various model architectures.

This is visible from the $88\%$ accuracy from the seen-seen test set and $95\%$ on the seen-unseen one. However, the network still performs poor on unseen classes.

Both siamese networks perform decently well on unseen test sets, with the larger embedding size outperforming the smaller one. A possible reason for this might be that the larger embeddings can better capture the underlying feature space of the data set and the object features like edges, corners, or color attributes. Thus, even for unseen classes these base properties helps it distinguish between similar and dissimilar images. However, both the siamese pipelines perform poorly on the seen classes compared to the classification based models. This is because optimizing triplet loss over features is a more challenging task compared to learning class labels. Since the dataset images are more geared towards single object classification, it is easier to learn the classification labels compared to reducing feature distances. However, siamese networks are more generalizable because they can handle any variety of objects or images. This is apparent from the unseen-unseen performance, while classifier networks fail to generalize outside of the train data.

Lastly, our custom end-to-end binary classifier network brings together both classifier and siamese approaches for a simpler and good performing framework. Since our task is just to classify between similar or dissimilar images, instead of learning distance between features we can easily learn to distinguish those features. Siamese networks learn to map feature vectors onto a latent space based on their relative distances which is quite difficult. Instead, by combining the input features and performing a binary classification we are just learning to draw a line between the two inputs and separate them based on their attributes. This is a much more easier task to solve using binary crossentropy. Our classifier achieves the highest performance on both seen-seen and seen-unseen datasets, and also has a decent $75\%$ accuracy on the unseen-unseen case which is comparable to the siamese networks.

A conspicuous observation from the test results is that seen-unseen classification performance for image matching tends to be higher than the seen-seen ones. This might be because it is easier to identify whether two images are dissimilar than to say if they are the same. To be declared same two images must have similarity across a broad range of attributes including subject, background, orientation, etc. But if even one of the features don't match they can be declared dissimilar. Since the model has definitely seen one of the samples, it can easily distinguish between the one it had not seen. Thus, it is apparent that seen-unseen matching is a much easier task.

## 6. Conclusion

The project investigated the topic of image similarity matching. Given two images, we had to determine whether they were similar or dissimilar. In order to accomplish this, we experimented with a number of deep learning architectures based on multiclass classification, feature mapping, and our custom end-to-end classifier. Our study on the Tiny Imagenet dataset demonstrates that siamese networks can better represent image features and are more generalizable for real world image matching tasks. Furthermore, our proposed model was capable of combining the advantages of end-to-end classifiers and siamese networks to achieve comparable or even superior performance on the same test.

## References

[1] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

[2] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[3] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? *Advances in neural information processing systems*, 32, 2019.

[4] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[5] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[6] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.