# Image similarity network: a mini project

January 21, 2020

**Abstract**

This work concerns creating a framework for classifying image similarity. Given two images, we first extract features using trained and modified ResNet50 and VGG16 convolutional neural network architectures. We then train a fully-connected neural network on the absolute difference of the two extracted features, to predict similarity. Competitive results are achieved on the Tiny ImageNet dataset, of **87.5%** accuracy on seen and **73.8%** on unseen object classes.

## 1 Introduction

Image similarity classifiers are an important step in computer vision's bid to improve AI's understanding of the objects and situations happening on images. Since lack of labeled data prevents neural networks from classifying *all* kinds of objects, it is an important task to train networks that are able to spot structural and conceptual differences of observed objects, just like humans do. In this paper a basic 2-step siamese similarity framework is introduced. The first step involves training image embeddings using convolutional neural networks. Then given two image embeddings, a 2 hidden layer neural network is trained to predict image similarity.

Convolutional neural networks have demonstrated great success on tasks like image classification and segmentation, with Image-Net being one of the most widely adopted model evaluation datasets in the computer vision community. Our similarity framework will be trained and tested on the Tiny Image-Net dataset.

## 2 Data

The Tiny Image-Net dataset is a subsample of the original Image-Net, with 200 distinct classes. The data originally comes with 3 folders, `train`, `val` and `test`, containing images for training, validation and testing respectively. However, the `test` folder does not have specified labels per image, which makes it impossible to test performance on it and hence data from it will be omitted in this work.

The images are all 64x64 pixels and contain 3 RGB channels. The pre-processing of the data consists of three steps. First, the images are brought onto a $[0, 1]$ scale by dividing all the channels by 255. Next, the images are normalized, by taking away the mean and dividing by the standard deviation of pixel values, across the training set $\mathcal{T}_{180}$ (the set is described in detail in 4.1). Lastly, the images are resized to 224x224, in order to be big enough to reasonably fit into the proposed classification architectures.

## 3 Model architecture

A siamese-style architecture will be used for the final similarity model. It differs from a regular siamese neural network by the clear separation of two training steps. First, a multi-class classification model $\mathcal{M}_c$ is trained to predict the label of each image. Secondly, the trained $\mathcal{M}_c$ will be used to extract features $x_1$ and $x_2$ from two images $I_1$ and $I_2$. Then, the absolute distance of the two features, $|x_1 - x_2|$, will be fed as input to a similarity model $\mathcal{M}_s$. The proposed image similarity framework is summarised on Figure 1.

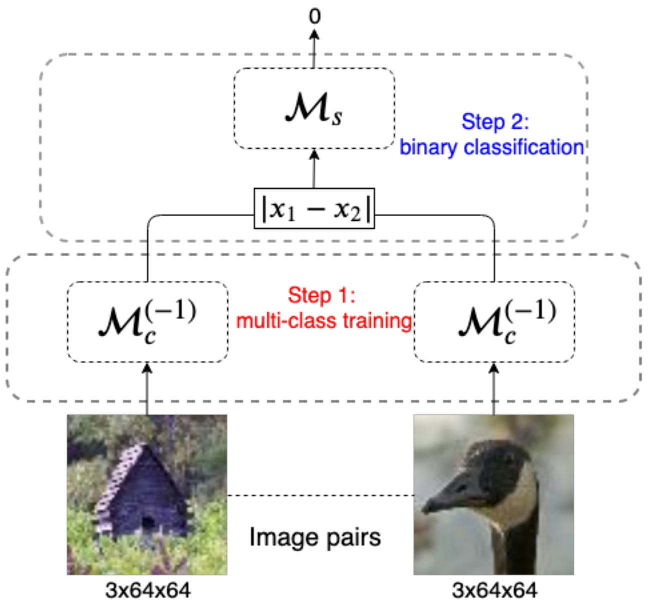


Figure 1: The proposed siamese-style similarity network.

Feature extraction on images using $\mathcal{M}_c$ is performed by taking the trained $\mathcal{M}_c$ and removing its last softmax layer (layer responsible for making class predictions), to give $\mathcal{M}_c^{(-1)}$. The image embeddings $x$ are taken to be the outputs of the newly created network $\mathcal{M}_c^{(-1)}$. This is motivated by the fact that up until the last softmax layer, the trained model already extracts the relevant information from the image to be able to do classification. Hence, the output of the layer before softmax is particularly informative in terms of features and structure existing in the image.

### 3.1 Multi-class classification models $\mathcal{M}_c$

The first step of training the similarity framework is multi-class classification. It can be thought of as trying to fit a model to create the best possible image embedding. It seems reasonable to assume that the better $\mathcal{M}_c$ is at classifying images, the more informative features we can obtain from the images and hence the $\mathcal{M}_s$ network can perform better at classifying similarity. Two model architectures for $\mathcal{M}_c$, that have proven to be very successful on the original ImageNet challenge, will be compared: ResNet50[1] and VGG16[2].

**ResNet50**

The state of the art image classifiers all use deep convolutional layer architectures, in order to be able to extract features at multiple scales in images. The deeper the networks, the richer and more informative the features that the model obtains, leading to better classification accuracy. However, the deeper the network, the more prone it is to the problem of vanishing gradients and degradation, meaning that in deeper networks the accuracy starts to saturate and ends up lower than

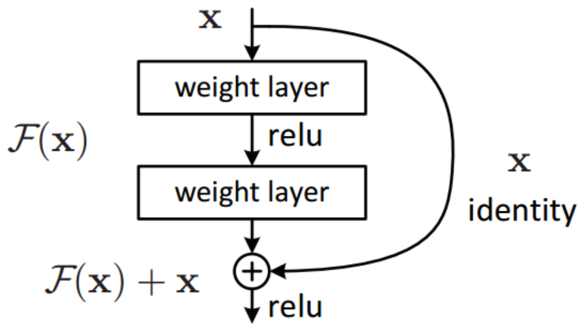

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2: The 'skip-connection' - a building block of ResNet.

in shallower networks. One solution to that problem is to use residual networks as proposed by K.He et.al [1]. A residual network like ResNet resembles a normal feed-forward network, but contains 'skip-connections', meaning that the input to some layers is saved and added to the input of a layer deeper in the network. Figure 2 illustrates the structure of a 'skip-connection'. All the activations in ResNet are ReLU, except for the last softmax layer. The weight layers in ResNet are stacked 3x3 convolutional and max pool layers. As we go deeper in the network, we gradually increase the number of convolution filters, starting with 64 and finishing with 256. The last two layers of ResNet are average pooling and a dense softmax layer, for classification. Here, ResNet50 will be used, which has 50 layers (including the average pooling and softmax output) and a total of 24 'skip-connections'.

The ResNet50 model considered here has 23,876,852 trainable parameters.

**VGG16**

The VGG architecture was proposed by A. Zisserman et.al [2]. It features a very traditional and intuitive stacking of convolutional layers with a small receptive field (3x3 kernels). Max-pooling is performed over 2x2 pixel windows, halving the size of the image.

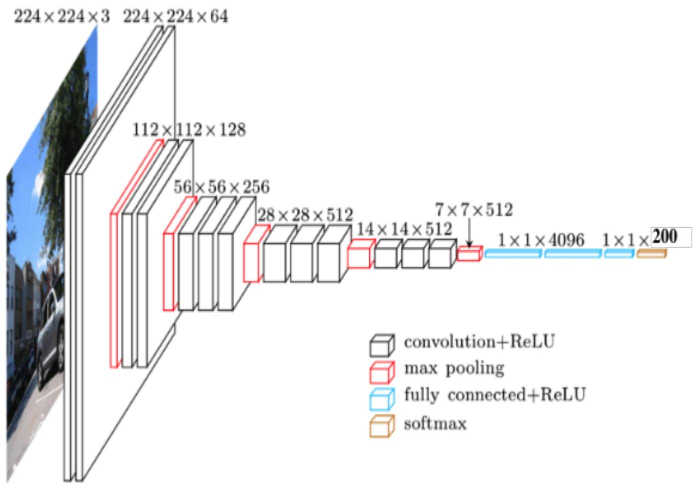


Figure 3: A simplified VGG16 architecture [3].

The VGG16 model has 16 layers and its simplified diagram is shown in Figure 3. The VGG16 used here has an additional average pooling layer before the fully connected layers. By stacking several 3x3 kernels at every convolutional layer, the effective receptive field is increased and the image has passed through many rectification activations, making the decision function more discriminative in comparison to using bigger kernels. The VGG16 used in this work has 134,998,004 trainable parameters.

**Feature extraction**

To obtain the feature models $\mathcal{M}_c^{(-1)}$, the last softmax fully connected layers are removed from the trained VGG16 and ResNet50 models. This results in embedding size of 4096 for VGG16 and 2048 for ResNet50.

### 3.2 Similarity model $\mathcal{M}_s$

A simple 2-hidden layer model is considered for the similarity classification. A 2-layer neural network has been chosen for $\mathcal{M}_s$ because since our feature space is quite high (4096 and 2048 for VGG16 and ResNet50 respectively), only a 2-layer or more network is guaranteed to find any arbitrary decision boundary that might be present in the absolute difference of image feature vectors [4]. After considering many architectures with varying hidden layer sizes, the final model was chosen to have half the number of units at each subsequent layer (except for the last layer which has 2 units). The diagram of the model is shown in Figure 4.

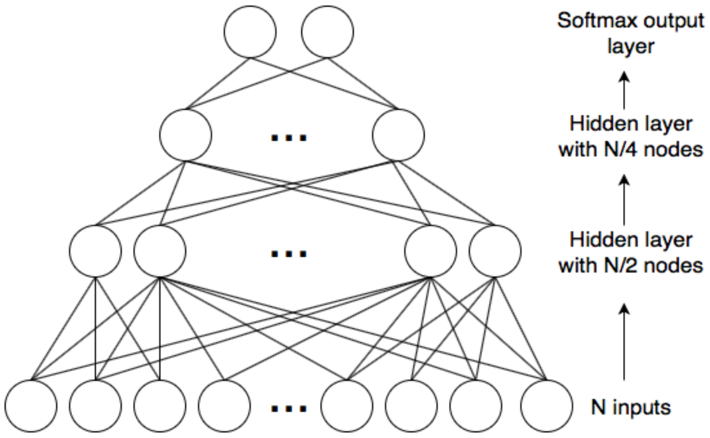


Figure 4: The similarity model.

A 25% dropout is added to all the hidden layers to prevent the model from overfitting. All weights in the model are initialized with the He initialization [5].

## 4 Training, validation and testing

### 4.1 $\mathcal{M}_c$ network

Since we are also interested to check $\mathcal{M}_s$'s performance on unseen categories, training of the feature extraction network will be performed on 180 out of 200 classes. This dataset will be taken from the `train` folder and the 180 classes taken arbitrarily. Let this dataset be called $\mathcal{T}_{180}$. In this way, we will have 20 classes that the model has never seen before. For training, a random 70-30 split of $\mathcal{T}_{180}$ for training and validation will be used, which means that we train on 63k images and validate on 17k. In this way we will have around 350 distinct images per class to train the network. This is not enough to robustly train a classifier without having it to overfit, and hence data augmentation will be used, described in more detail in 3.3. After training, the model's classification performance will be tested on the images contained in the `val` folder (from the appropriate classes that we trained on), which have 50 image examples per class.

### 4.2 $\mathcal{M}_s$ network

The training and validation sets of the similarity network are not immediately obvious. In principle, from the 90k images belonging to $\mathcal{T}_{180}$, we could form around 8 billion distinct pairs of images. This is too much training data for our model to digest, and would result in the dataset being highly imbalanced (as most pairs would be dissimilar). However, it seems wasteful to just pick a fixed number of image pairs and train only on them, with all this abundancy of 'free' new examples. There-

fore, at each new epoch, we will sample 644,400 image pairs from $\mathcal{T}_{180}$ in a manner that we have a 50-50 split between similar examples and dissimilar examples. This will be done in a balanced manner, so that we roughly have the same class mixtures in the pairs. It means that the sampled dataset will contain 10 image pairs $(I_i, I_j)$, where $I_i \in \text{class}_i$ and $I_j \in \text{class}_j$ and $i \neq j$, and we let $i, j = 1 \rightarrow 180$. To balance it out we also add 1790 image pairs $(I_i, I_i)$, where $i = 1 \rightarrow 180$.
The validation dataset containing 386,640 image pairs, is sampled in the same way as described above at the beginning of training and before each epoch, image pairs present in the validation set are removed from the new random training set.

### 4.3 Data Augmentation

Since we only have 350 images per class to train the $\mathcal{M}_c$ network, we need to augment our data in order not to overfit the model. Data augmentation helps the model to be more robust to slight changes in the image, making it more generalisable. I will consider 4 types of image transformation. The augmentation process is applied to every loaded batch in each epoch and consists of each of the 4 types being applied randomly with probability 0.5 to every image in the batch.

- **Random color jitter** - randomly jitter an image's brightness, contrast, saturation and hue. Brightness, contrast and saturation jitter factors are sampled uniformly from [0.9, 1.1] and hue factor is sampled uniformly from [-0.1,0.1].
- **Random crop** - randomly cropping the image to a 56x56 size. This means that a 56x56 window is chosen randomly in the image, and only the pixels in the window retained. Since in the Tiny ImageNet images the class object is always located centrally in the picture, the model should still understand the characteristics of that object if we crop out a small part of the picture.
- **Horizontal flip** - flipping an image from left to right, changing the order of the pixels.
- **Random rotation** - the image is randomly rotated by an angle $\theta \in [-\pi/3, \pi/3]$. The cropped out parts of the image are filled with 0's.

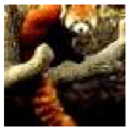
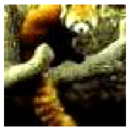
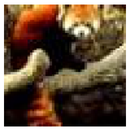
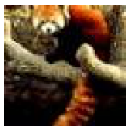
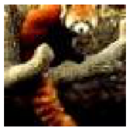
| Original | Jitter | Crop | Flip | Rotation |
|---|---|---|---|---|
|  |  |  |  |  |
| | | | | |
| | | | | |

Table 1: Illustration of different image augmentation transforms.

We can see in Table 1, that it is not too hard to classify the correct object on the augmented images i.e. a human would not be fooled by such transformations. This further motivates the use of data augmentation, as it makes the model understand the underlying features of the class objects better. By transforming each image in a batch with the above process, we effectively increase the number of examples per class from 350 to $350 \times 2^4 = 5600$.

### 4.4 Optimizer

All training is performed with the Nesterov's Accelerated Gradient (NAG) optimizer. For $\boldsymbol{\theta}_t$ the weights of the model and $\boldsymbol{v}_t = \Delta\boldsymbol{\theta}_t = \boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}$ the update at iteration $t$, the weight update is performed as follows:

$$\begin{aligned} \boldsymbol{u}_t &= \gamma \boldsymbol{u}_{t-1} + \eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t - \gamma \boldsymbol{u}_{t-1}) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \boldsymbol{u}_t \end{aligned}$$

where $\boldsymbol{u}_0 = 0$ and $L$ is the loss function, in this work always cross-entropy. The $\gamma$ momentum parameter, controlling the amount with which the optimizer 'looks to the future', is set at 0.8 for all training. In this work NAG optimizer turned out to be superior to traditional SGD or SGD with momentum. Because updates in consecutive iterations tend to be the same, $L(\boldsymbol{\theta}_t - \gamma \boldsymbol{u}_{t-1})$ approximates the loss at the next time-step. Thanks to that, NAG can essentially prevent a big update in a direction where the loss quickly slopes back up again.

### 4.5 Learning rate decay

The same learning rate decay schedule will be applied in training both models. Decaying the learning rate helps the model to stabilize learning from the noisy high-variance mini-batches when the model already reaches a local minimum of the weight space. Having a big learning rate in that part of training can lead the model to overshoot the local minimum, making it stuck between loss ravines. It is therefore good practice to start training with a relatively high learning rate and reduce it as the validation accuracy begins to plateau.

The decision whether to decay the learning rate is made after each epoch (greater than third). Let $V^{(k)} = \{v_1, v_2, ..., v_k\}$ be the ordered set of validation accuracies achieved after each epoch and let $\mathbb{E}[V^{(k)}] = \frac{1}{k}\sum_{i=0}^{k} v_i$. Learning rate is multiplied by 0.5 if the following holds:

$$\mathbb{E}[V^{(k)}] - \frac{1}{3}(\mathbb{E}[V^{(k-1)}] + \mathbb{E}[V^{(k-2)}] + \mathbb{E}[V^{(k-3)}]) < \alpha$$

which intuitively means that we reduce the learning rate if the average difference of the current mean of validation accuracies with means of validation accuracies from previous 3 epochs is less than $\alpha$. We use $\alpha = 0.02$ for $\mathcal{M}_c$ training and $\alpha = 0.01$ for $\mathcal{M}_s$ training. Additionally, we restrict that after a learning rate decay, we must wait 3 epochs to perform another one.

## 5 Experiments

All experiments were run on a NVIDIA Tesla K80 GPU. Using a GPU decreased the training time and evaluation of both models by 50 fold.

### 5.1 $\mathcal{M}_c$ network

The weights of both ResNet50 and VGG16 have been initialized to the trained weights of these models used for the ImageNet classification in their original papers [1,2]. This way, training time was greatly reduced as the trained weights provide a good initial guess as to where the best parameters sit in the weight space. Even though we are using pretrained weights, it is still necessary to train the networks, as the downsized and then upsized images are very different from the original 256x256 ImageNet images used for their training. Both models were trained with mini-batch of 64 and initial learning rate of 0.01. This means that there were 985 weight updates during a epoch. Figure 5 shows the progression of training for both models. The grey dashed lines represent moments when the learning rate was decayed by half.

We can see that the learning rate decay was always accompanied by a relatively substantial increase in validation accuracy. We can also see that ResNet50 outperforms the VGG16 by almost 5% in validation accuracy. Both models slightly overfit the data, but as it comes with no decrease in validation accuracy, its not a big concern.

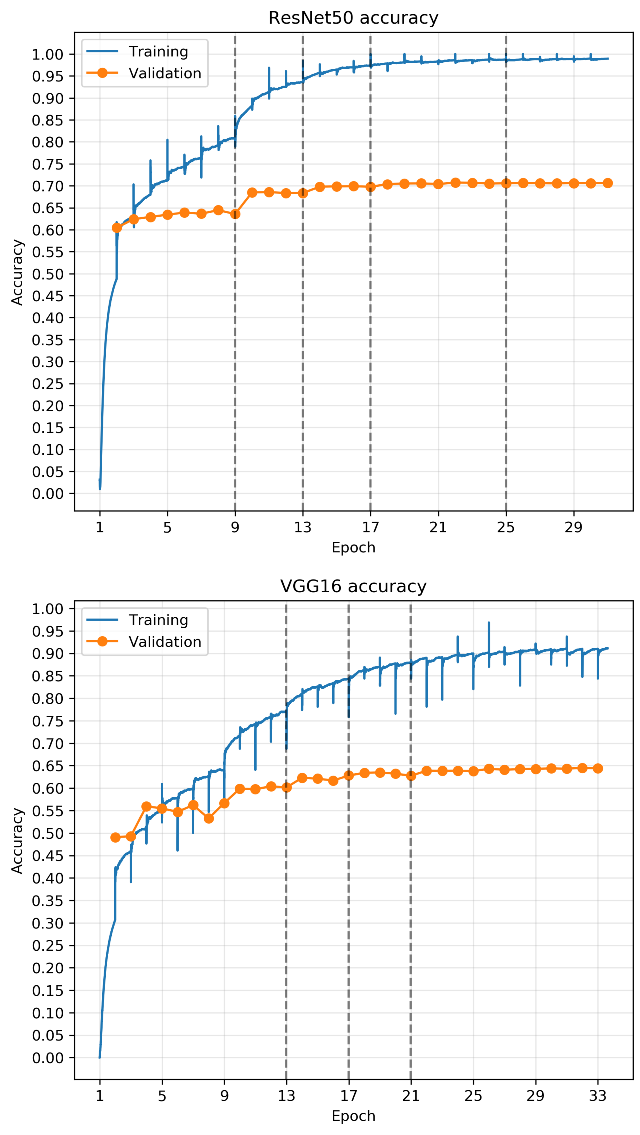


Figure 5: Multi-class accuracy during training of ResNet50 and VGG16.

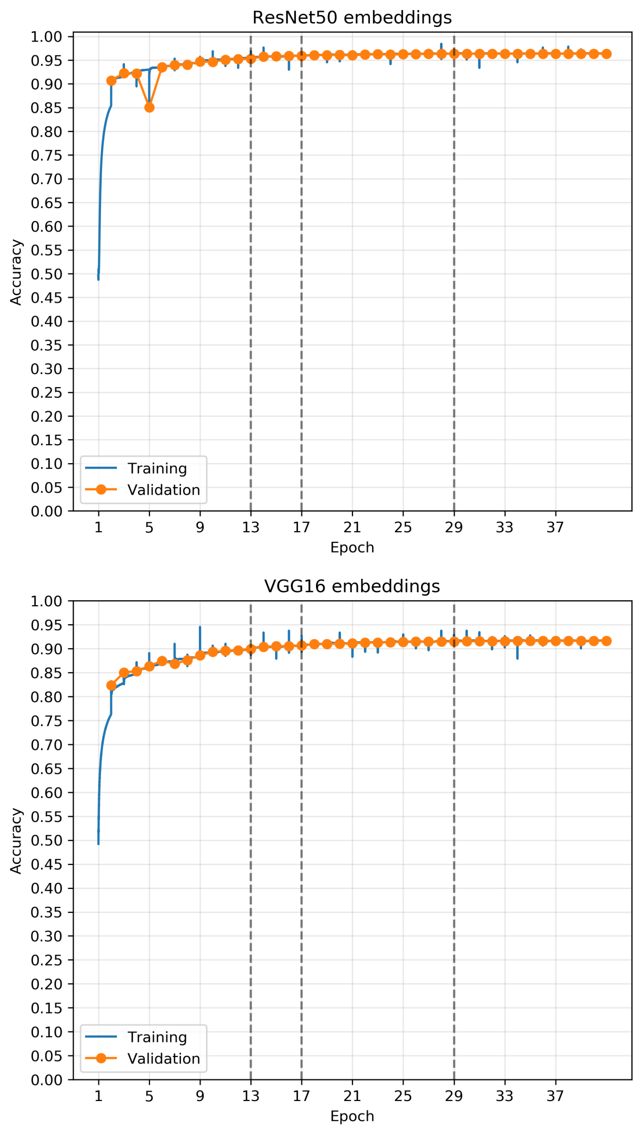


Figure 6: Accuracy of similarity classifier during training, using embeddings from ResNet50 and VGG16.

### 5.2 $\mathcal{M}_s$ network

The similarity network is trained using a mini-batch of 256 and learning rate 0.005. This corresponds to around 2500 weight updates per batch (not exactly because at each epoch the training set is sampled randomly and pairs existing in validation set removed). Using a larger mini-batch makes the gradient updates a lot less noisy and hence the models learn very quickly. Since we sample a new training set at every new epoch, the model does not overfit one training set and the validation accuracy follows closely the training accuracy, which is visible for both types of embedding in Figure 6.

We can see that the ResNet50 embeddings achieve a 5% better validation accuracy than the VGG16 embeddings, which is consistent with the former being a better multi-class classification model. The best validation accuracy is achieved by the ResNet50 embeddings, giving **96.3%** at the 33rd epoch.

## 6 Evaluation

The accuracies of the ResNet50 and VGG16 multi-class classifiers on the test set are **69.51%** and **67.96%** respectively. Since this is comparable to the achieved validation accuracies, we conclude that overfitting did not occur. This is partially due to using data augmentation, the dropout layers present in both models and learning rate decay.

The test set for $\mathcal{M}_s$ is composed in the same way as described in 4.2, but from data in the `val` folder, unseen during training of both $\mathcal{M}_c$ and $\mathcal{M}_s$. The test set contains a total of 3,980,000 (!) distinct image pairs. The accuracies of $\mathcal{M}_s$ using ResNet50 and VGG16 image embeddings are **86.51%** and **82.54%** respectively. At first sight it looks like the validation accuracies of $\mathcal{M}_s$ are a lot higher than testing accuracies. However, we have to remember that in the validation data, the image pairs come from $\mathcal{T}_{180}$ (section 4.1), meaning that all the classes have been previously seen by the embedding model $\mathcal{M}_c$. To test performance on unseen and seen categories we recognize five cases:

- **Case 1**: similar images and classes of both images seen in training data.
- **Case 2**: dissimilar images and classes of both images seen in training data.
- **Case 3**: dissimilar images and 1 class seen in training data and other unseen.
- **Case 4**: similar images and classes of both images unseen in training data.
- **Case 5**: dissimilar images and classes of both images unseen in training data.

| Case | ResNet50 features accuracy | VGG16 features accuracy |
|---|---|---|
| 1 | 83.36% | 83.40% |
| 2 | 92.22% | 83.85% |
| 3 | 84.27% | 73.47% |
| 4 | 74.25% | 83.00% |
| 5 | 69.56% | 56.89% |

Table 2: Accuracies of $\mathcal{M}_s$ on test set.

We can see that both the ResNet50 and VGG16 feature accuracies in cases 1 & 2 are a lot lower than the validation accuracies (drop from 96.3%/91.2% to 87.55%/83.61% for ResNet50/VGG16). This means that the final $\mathcal{M}_s$ models are overfit on the training image pairs. It is interesting to note that both $\mathcal{M}_s$ models struggle in case 5 the most - meaning that they find it harder to say that images from unseen categories are dissimilar.

Since a binary classification greatly depends on the probability threshold above which we predict a certain class, it is useful to look at the Receiver Operating Characteristic (ROC) curve. This curve is produced by counting the total number of false positive and true positives for every possible probability threshold. On Figure 7 we can see the ROC curves plotted for both feature embeddings and on various subsets of the test set. The red line indicates the ROC for the entire test set, yellow for seen examples (cases 1 and 2) and green for unseen examples (cases 4 and 5).

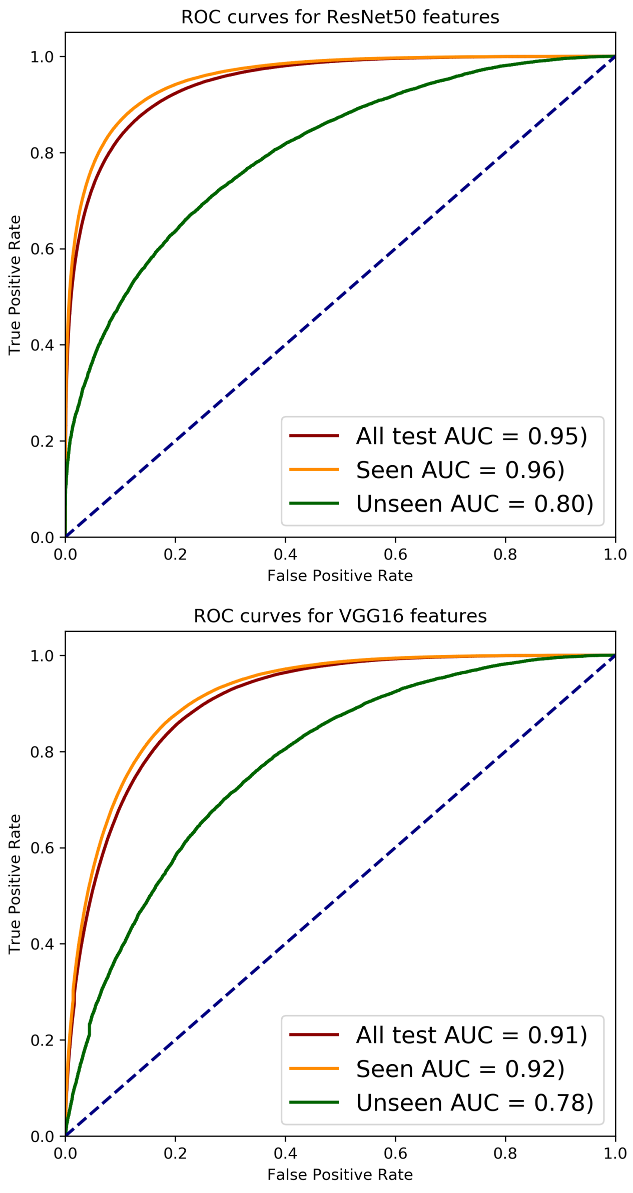


Figure 7: ROC curves of similarity classifier on test data, using embeddings from ResNet50 and VGG16.

## 7 Discussion

It appears that the $\mathcal{M}_s$ are slightly overfitted on the training data. To counter that, a bigger proportion of dropout could be used on the fully connected layers and a weight decay penalty could be applied to the loss function.

For future implementation it would be interesting to explore an ensemble model from both the ResNet50 and VGG16 features, as looking at Table 2, it seems that VGG16 is better at classifying similar images from unseen categories.

In this work, we have used the L1 distance (absolute distance) between the two images feature vectors. That was an arbitrary choice and future implementations could explore using different distance measures, like L2 or even train a neural network with combined features as input.

## References

[1] *Deep Residual Learning for Image Recognition*, K. He et.al, 2015

[2] *Very Deep Convolutional Networks For Large-Scale Image Recognition*, A. Zisserman et.al, 2015

[3] *Accelerating Deep Neural Networks on Low Power Heterogeneous Architectures*, J. Cano, 2018

[4] *Decision Boundary Formation of Neural Networks*, C. Lee et.al, 2014

[5] *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, K. He et.al, 2015