ECE4179: Semi-supervised learning

Nicholas Prowse
npro0001@student.monash.edu

28756401

Contents

1 Summary

This report shows a comparison of 3 different techniques of semi-supervised learning. Semi-supervised learning is a field of machine learning that allows a model to gain high accuracy with a very small set of labelled training data, and a large unlabelled dataset. This is a very important field of study as labelling data is time consuming and expensive, while unlabelled data is very easy to obtain. The 3 different methods I compared are

- Rotations: Each unlabelled image is rotated by a multiple of 90° and a model is trained to recognise the rotation that was performed. Transfer learning is then used to train the network on the labelled data.
- Relative Patch Location: Each unlabelled image is split into 9 small patches, and the centre patch along with one other random patch are chosen. The model is trained to predict which patch was chosen. Again, transfer learning is then used to train this network on the labelled data.
- Solving Jigsaws: Again, each unlabelled image is split into 9, but this time all the patches are jumbled up. The model is trained to predict the permutation used to jumble up the image. Finally, transfer learning is used to train this network on the labelled data.

All three of these methods provided far better accuracy than what can be achieved with just the labelled training data, with the solving jigsaw task providing the best performance of the 3.

2 Introduction

Traditionally, machine learning techniques have relied on large labelled datasets in order to create models that accurately learn visual representations. In the modern information age, there is no shortage of data, however, labelling the data is very time consuming and expensive, as a human is required to individually label each data point. As such, recently there has been a lot of research into semi-supervised learning techniques. Semi-supervised learning involves training a model using a very large unlabelled dataset, and a small labelled dataset. The model is able to learn the statistics and features of the images from the unlabelled dataset, and learns the labels and classes from the labelled dataset.

In order to achieve this, semi-supervised techniques typically rely on one of three assumptions. The continuity assumption assumes that similar images will share a label, meaning that simple decision boundaries should be able to distinguish the classes[1]. The clustering assumption assumes that images will form discrete clusters, and images in the same cluster are more likely to share the same label. This is similar to the continuity assumption, however the continuity assumption doesn't assume the data is clustered, it just assumes that images close to each other will have the same label[1]. Finally, the manifold assumption assumes that images of the same class lie on a manifold with much lower dimensionality than the input. For example, if the input was 3 dimensional, then data points from one class may lie approximately in a straight line[1]. Provided the dataset satisfies one of these three properties, then semi-supervised learning should be possible, as we can create algorithms which find these statistical features of each class.

There are many different techniques that achieve this, however, in this report I will be focusing on two stage techniques. These consist of two stages: in the first stage, the network is trained on the unlabelled dataset using self supervised learning. In the self supervised learning stage, we train the network to perform a pretext task. Pretext tasks are chosen in such a way that in order for the network to perform them it must learn the structure and features of the objects in the image. Once the network has completed self supervised learning, we move onto the second stage, where we use transfer learning to train the network on the labelled data. The last layer of the network is removed, and replaced with a new layer to fit the classification task, then just this layer is trained to classify the labelled data. The network should already be able to extract features of the image, so this training should be able to reach a higher accuracy that if the network was trained solely on the training data.

I will be exploring 3 techniques to achieve the self supervised learning stage. The first involves rotating each image by one of $0°, 90°, 180°$ or $270°$, and training the network to detect the rotation that was performed on the image. The second involves the network learning the relative location of two patches of the image. Two small adjacent patches of the image are passed into the network, and the network is trained to detect the relative location of the second patch compared to the first. For example, the second patch may be on the top, top left, bottom right etc. The third technique involves training the network to solve a jigsaw of the image. The image is split into 9 patches, randomly permuted, and the network is trained to predict what the permutation was. In order for a model to be able to perform any one of these tasks, it must have an understanding of the general structure of the images. For example, in order to recognise the rotation of an image, the model

will have to learn that typically sky is at the top of an image, eyes are above noses, legs go underneath the body, etc. Since it must learn to recognise these features, it should be easily transferable to the task of classification, even if the data set is relatively small. In order to compare the 3 techniques, we will use the ResNet18[2] model to train the classifier. The final layer is replaced with a fully connected layer that outputs 10 classes, but otherwise I have not changed the model. The model will be trained using the STL10 dataset, which is a subset of ImageNet designed for semi-supervised learning[3]. It contains 100,000 unlabelled images, along with 5000 training images and 8000 validation images. However, I will just be using 500 training images, and the other 4500 will be testing images. The images are 96×96 and are labelled into 10 classes. The unlabelled set contains a broader distribution of images that the training or testing images. It contains animals and vehicles that are not in the training or validation set[3]. This is to mimic a real world scenario, since if we are using unlabelled data, then that typically means that each individual image hasn't been examined in detail. Thus, we would typically expect there to be other types of images in an unlabelled set.

3 Baseline

In order to create a baseline to compare to, I trained a ResNet18 model using just the training data. I used a learning rate of 10^{-3}, and halved it every 25 epochs. The model was trained for 100 epochs and achieved a test accuracy of 46.5%. Figure 1 shows the loss and accuracy during training. We can see that the training accuracy gets very high (99.8%), but the accuracy on the test dataset is much lower. This is due to the small training dataset, which allows the network to essentially memorise the training set. We can see this in the loss plot, where the validation loss does not improve at all beyond the first few epochs, while the training loss is constantly decreasing

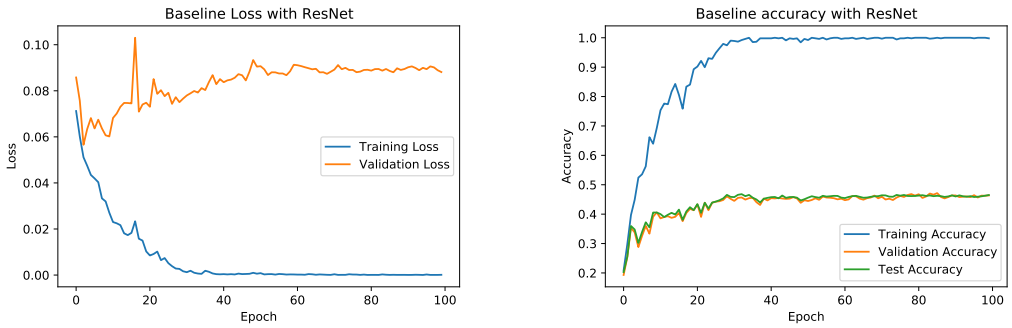


Figure 1: Accuracy and loss of the model when trained on 500 labelled training images

4 Rotations

In this section we will explore how we can use image rotations to train a neural network to extract visual representations of an image. Each image is rotated by some multiple of 90°,

and the network is trained with the rotation amount as the label. The final layer of the ResNet18 is replaced with a fully connected layer that outputs 4 classes, corresponding to the 4 different rotations. The idea is that the network cannot determine the rotation that has been performed if it cannot recognise the visual features of the images. For example, if the dataset contained faces, then it would likely learn to recognise eyes, noses and mouths, and then use their locations relative to each other to determine the orientation of the face[4]. Once this is done, we replace the final layer with a fully connected layer that has 10 outputs, and train the network to classify the STL10 images. The learned visual representation from training on the rotations should improve the performance of the model.

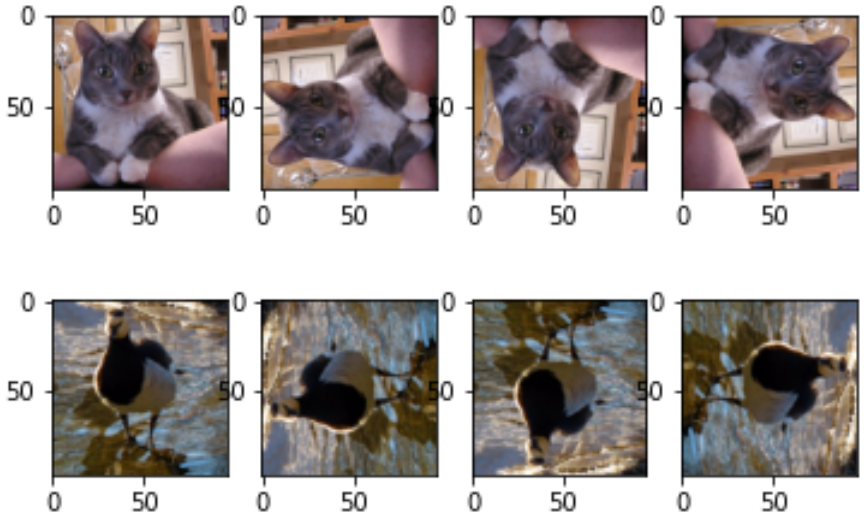


Figure 2: Example of how two different images will be transformed. The network is trained to detect which of these 4 rotations were performed

The rotations are performed using vertical flips and transposes, as these are more computationally efficient than using PyTorch's rotate transform. Transposes and flips also don't introduce any visual artefacts, whereas, depending on the implementation, rotations can introduce visual artefacts that the network may take advantage of. To prevent this we use the transformations proposed by Gidaris et al.[4].

- 0°: Do nothing
- 90°: First transpose, then flip vertically
- 180°: Flip vertically
- 270°: First flip vertically, then transpose

To perform a 180° rotation, we should really flip horizontally and vertically, but since we are using random horizontal flips as data augmentation, preforming a horizontal flip to rotate the image would be redundant.

Each image has 4 different rotations, which means that with 100,000 unlabelled images, our dataset has 400,000 training samples in it. The model was trained for 30 epochs using an initial learning rate of 5×10^{-3}. The learning rate was multiplied by 0.3 every 10 epochs. The model reached a final accuracy of 92.7% on the validation set, and took 5 hours to train.

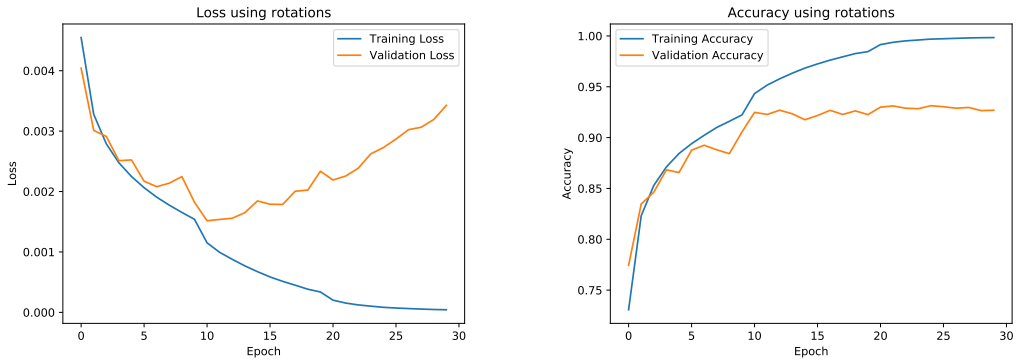


Figure 3: Accuracy and loss of the model at classifying the rotation applied to an image

Once the model had been trained to recognise rotations, the last fully connected layer was removed, and replaced with a new one with 10 outputs, corresponding to the 10 classes of STL10. It is expected that the network should be able to extract features of the images now, so to start with, we just train the final layer, with the rest of the network frozen. The last layer will use the features extracted by the convolutional layers to determine which class the image belongs to. The last layer was trained for 100 epochs with a learning rate of 2×10^{-2}, which was dropped down to 4×10^{-3} at epoch 50. Once the final layer was trained, I unfroze the rest of the network to fine tune the parameters. Here I used a lower learning rate of 5×10^{-3}, and dropped it down to 1×10^{-3} after 50 epochs. Performing the transfer learning in two steps like this provides better performance than training the whole network from the start. Since the network should already be able to extract features, training the last layer should allow the network to get relatively close to a good local optimum. Since the network is close to good parameters, when we fine tune the network it will fall into this local optimum and have a relatively good accuracy. However, if we start by training the entire network, the network will initially be further away from this good local optimum, so it is possible that it could fall into a different local optimum with worse performance.

Figure 4 shows the loss and accuracy of the network during the full training phase. The final test accuracy of the network was 55.0%, which is an 8.5% improvement over the baseline accuracy. We can see that there was a significant accuracy improvement after epoch 100, when the network was unfrozen. This indicates that the feature extraction learned by recognising rotations is not perfect, as the network gained significant improvement by fine tuning it. However, it was good enough to get a significant performance boost over the baseline accuracy.

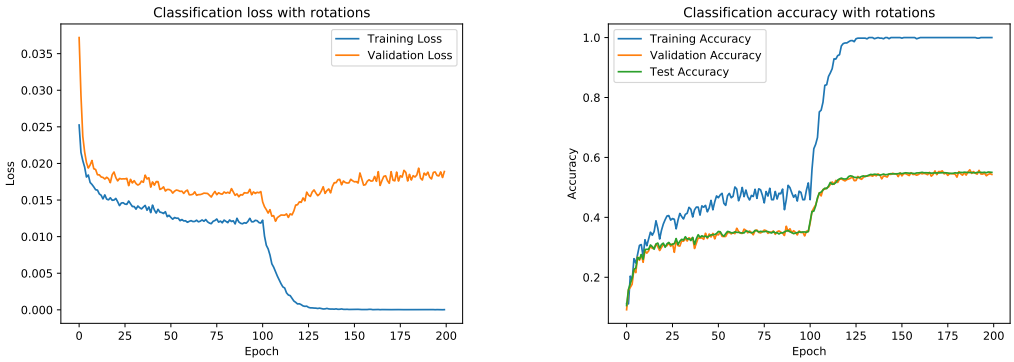


Figure 4: Accuracy and loss of the model at classifying the STL10 images, when trained with the pretext task of recognising applied rotations

5 Relative Patch Location

This technique involves splitting the image into 9 small patches, selecting the centre one, and one other at random, and training the model to predict which position the second patch was taken from. The idea here is that the network will learn the features of the images, and then use these features to determine where one patch would be relative to another. For example, if one patch has wheels in it, and the other has the body of a car, then the model would detect these features and decide that the wheels must be below the body of the car.

If we did this without any data transformations before hand, the network would not learn to extract features. Instead it would exploit low level features of the image such as boundary features and chromatic aberration. The boundary shapes and features could be used by the model to determine how the two patches fit together. This is not what we want, as it allows the model to learn how the patches are located, without learning the features of the image. To prevent this, we will perform a random crop on each patch, meaning that the two patches passed into the model will not line up perfectly, and there will typically be a gap between them. This prevents the edges of the patches from matching up, meaning the network cannot use the boundaries[5]. Another low level phenomena that the model could exploit is chromatic aberration. Due to the way camera lenses focus light, the brightness of colours throughout the images aren't uniform. Typically one colour will be brighter than the others in the sensor. For example, the green channel may be brighter, closer in to the middle of the image than the other channels. The model could use this fact the differentiate between corner and edge image patches. To prevent this, we randomly convert some of the images to grayscale with probability 2/3. This is done by averaging the 3 colour channels. This means that the model cannot rely on differences between the colour channels, but since some images are still colour, it can still learn information about colours of objects[6].

Before isolating the two patches, I perform a random horizontal flip followed random crop to size 90×90. It is important this crop doesn't have padding, as any padding could be used to identify the location of a patch. Then, I perform a centre crop to a size of 30×30 to get the centre patch, and a crop of one of the 8 surrounding 30×30 patches. The patch is selected at random. Then each patch is randomly cropped to size of 26×26 and

is converted to grayscale with probability 2/3. Finally, each patch is resized to 96×96, as this is the size of images that we want the classifier to work on.

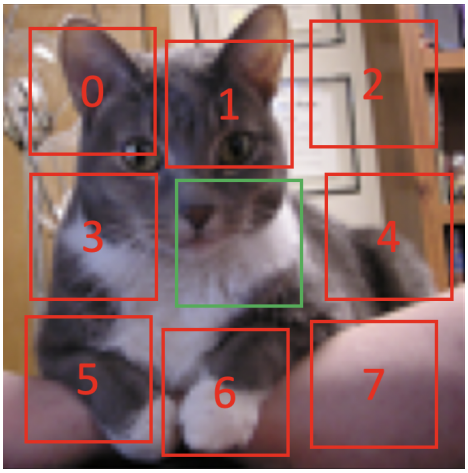


Figure 5: Example of how the two patches are created. The centre patch in green is always used. The second patch is randomly chosen from one of the other 8, and the network has to predict which one it is. To prevent boundary detection by the network, random jitter is applied to the location of each patch, which is achieved through a random crop

Our model will consist of a ResNet18, with its final fully connected layer removed. Thus, the output of this will be 512 dimensional. Both images are passed into the network separately, to obtain two 512 dimensional vectors. These are then concatenated and passed through a 1024×8 fully connected layer. Figure 6 shows this design

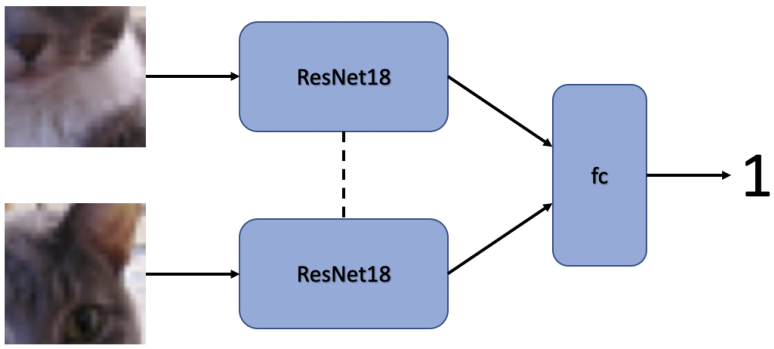


Figure 6: Design of the relative patch location network. The dotted line between the two ResNet's indicates they have the same weights (i.e. they are the same network). The ResNet has its final fully connected layer removed. In the example shown above, the second image should go directly above the first, so the network should output a 1

Since each image has 8 possible image patch options, the effective size of the unlabelled dataset is 800,000. This is very large and take a very long to to train each epoch, so

instead, I have left the dataset size at 100,000, and for each image, a random patch is chosen. The model was trained for 50 epochs with a learning rate of 0.02, which was halved every 20 epochs. Figure 7 shows the loss and accuracy of as it was trained. The model only managed to reach a validation accuracy of 36.3%, but this is a much more difficult task than the predicting rotations.

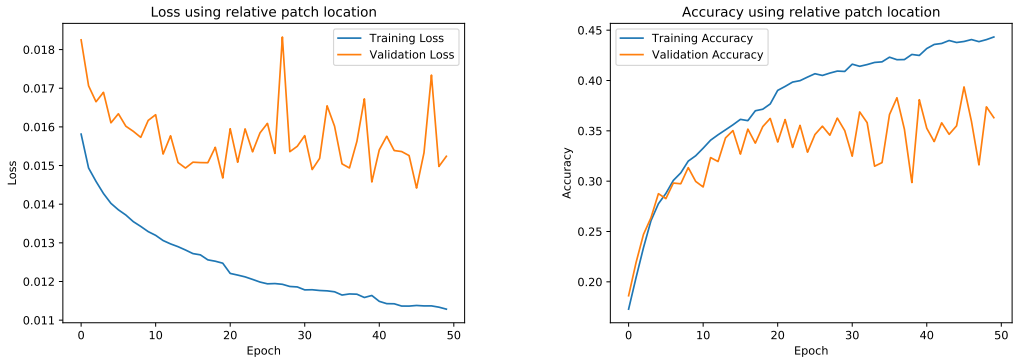


Figure 7: Accuracy and loss of the model at recognising the position of one patch of an image relative to another

After the model was trained on the pretext task, the ResNet is taken out of the model, the fully connected layer is added back to the end and the network is trained on the labelled data. The same approach that was used for the rotations task is used here, where we train just the last layer for the first 100 epochs, then we train the entire network for 100 epochs to fine tune the parameters. The final accuracy on the test dataset was 55.3% which is very similar to the accuracy achieved using rotations. However, here we can see that very little improvement occurs during the fine tuning stage (after 100 epochs). This means that the parameters found by training on the pretext task did not need to be changed much to transfer them over to the classification task. Thus, when training on predicting relative patch location the model learns to extract image features very well.

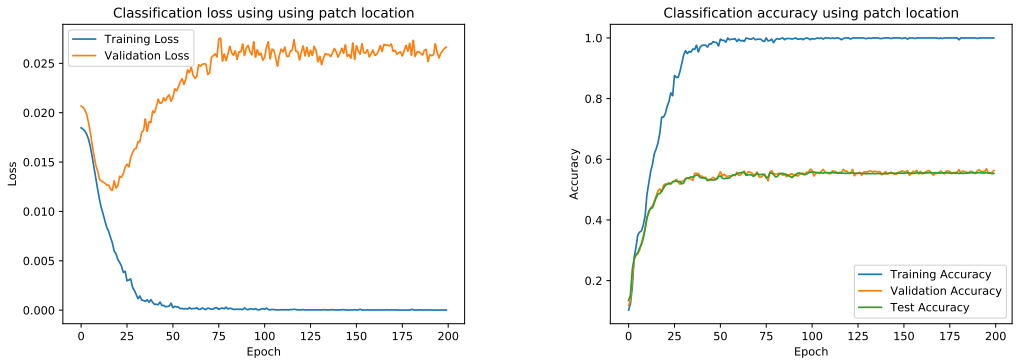


Figure 8: Accuracy and loss of the model at classifying the STL10 images, when trained with the pretext task of recognising relative patch locations

6 Solving Jigsaws

The final technique I have used is to train the network to recognise how patches of an image have been permuted, similar to solving a jigsaw. The image is split up into 9 pieces, randomly permuted, and the network must output the permutation that was used. The concept here is similar to that used for the relative patch location. In order for the network to recognise where each patch should go it must be able to recognise high level features in the image. Similar to when we used relative patch location, we must perform a few transformations on the patches so that the network can't exploit low level features of each patch. Namely, we randomly jitter the location of the patches, to prevent the edges from lining up perfectly, and we randomly convert to grayscale with probability 2/3 to prevent the network learning about chromatic aberration. As such, we will do exactly the same pre-processing on the images and patches as we did in the previous section.

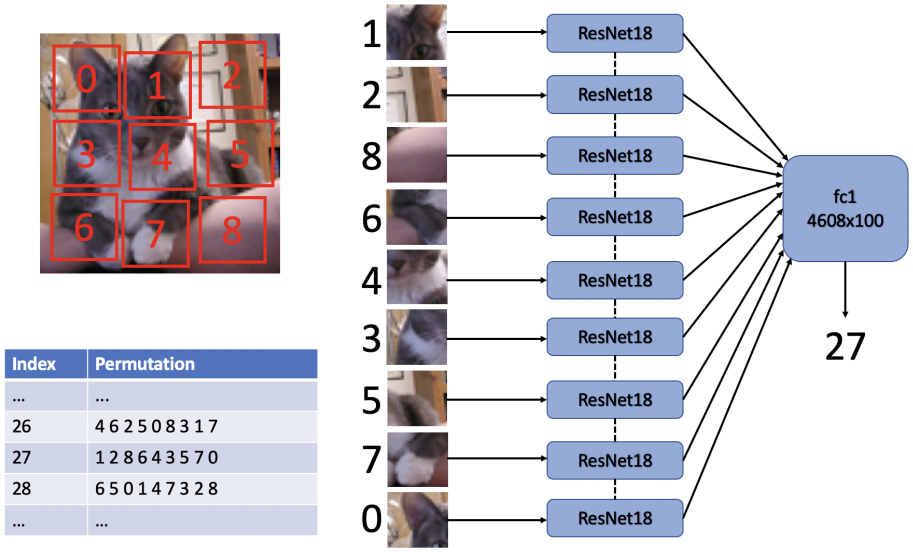
There are $9! = 362880$ different possible permutations, but it would be infeasible for the network to have this many outputs. As such, rather than using any random permutations, I will use a predefined set of 100 permutations, and the model only has to choose from those 100. However, if the permutations are two similar, the network may struggle to distinguish them. For example, if one permutation is the same as another, except two patches are swapped, the network will often get them mixed up[7]. As such, when I generate the set of permutations, I will ensure that they all have a Hamming distance of 3 or more, so that none of them are too similar. The Python code below was used to generate these permutations. It works by randomly selecting a permutation, then as long as the Hamming distance to all other permutations already found is not greater than 2, it continues randomly selecting a new random permutation. Once a permutation is found that has a Hamming distance greater than 2 to all permutations, it is added to the list of permutations. When this was done, I had a sequence of permutations with a minimum Hamming distance of 3, and an average Hamming distance of 8.003. This should ensure that no permutations are similar enough that they can't be distinguished.

```
import numpy as np
from scipy.spatial import distance

n = 100
perms = np.zeros((n, 9))
perms[0] = np.random.permutation(9)
for i in range(1, 100):
    next_perm = np.random.permutation(9)
    while min([distance.hamming(next_perm, p) for p in perms[0:i]]) <= 2/9:
        next_perm = np.random.permutation(9)
    perms[i] = next_perm

np.save('perms.npy', perms, allow_pickle=False)
```

Similar to the design used for the relative patch location, I will use a single ResNet18 with the final fully connected layer removed. Each of the 9 patches will be fed into the network separately, and the 9 outputs will all be fed into a fully connected layer. Each ResNet has a 512 dimensional output, meaning that the input to the fully connected layer is $512 \times 9 = 4608$. The output has dimensionality of 100, corresponding to the set of 100 permutations



Index	Permutation
...	...
26	4 6 2 5 0 8 3 1 7
27	1 2 8 6 4 3 5 7 0
28	6 5 0 1 4 7 3 2 8
...	...

Figure 9: Design of the jigsaw network. The dotted lines between the ResNet's indicates they have the same weights (i.e. they are the same network). The ResNet has its final fully connected layer removed. In the example shown above, the permutation corresponds to index 27 in the permutations list

The jigsaw solving model was trained for 30 epochs with an initial learning rate of 5×10^{-3}. The learning rate was halved every 10 epochs. In each epoch the network is trained once on each of the 100,000 images, and for each image a random permutation from the set of 100 is chosen. Figure 10 shows the loss and accuracy of this model as it trained. The model took 9 hours to train and managed to reach a very high accuracy of 84.7% on the validation dataset.

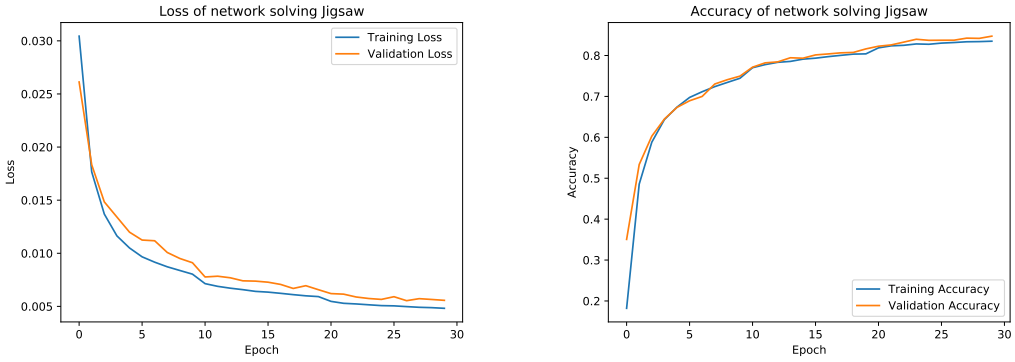


Figure 10: Accuracy and loss of model when trained to solve jigsaws. The final validation accuracy of the model was 84.7%

Once the above model was trained, the ResNet18 network is isolated, and a fully connected layer is added back to the end, with 10 outputs. As with the previous experiments, we freeze all layers except the last for the first 100 epochs, then we unfreeze all the layers

and fine tune the network. The first 100 epochs used a learning rate of 0.02 and the last 100 epochs used a learning rate of 5×10^{-4}. Figure 11 shows the loss and accuracy of the model during training. We can see that most of the improvement happens at the very start when we are only training the last layer, and very little improvement happens when fine tuning the entire network. This indicates that the pretext task of solving jigsaws allows the model to learn to extract the features of the images very well, meaning that not much fine tuning is required.

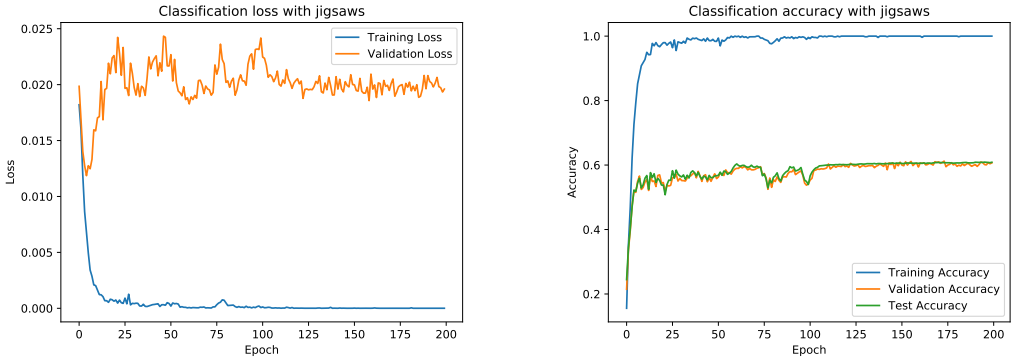


Figure 11: Accuracy and loss of model when trained with the pretext task of solving jigsaws

7 Comparison

Table 1: Accuracy on the test dataset, and train time of each of the methods

Method	Test Accuracy	Training time
Baseline	46.5%	15 minutes
Rotations	55.0%	5 hours
Patch Location	55.3%	3.5 hours
Jigsaws	60.9%	9 hours

Table 1 shows the accuracy and training time of the four different methods. All 3 semi-supervised techniques gave a significant improvement to the accuracy when compared to the baseline. Using rotations and patch locations had very similar accuracy results, with relative patch locations requiring less training time. However, we could probably get similar performance from the rotations method using fewer epochs, as the validation loss was not improving for the second half of training. Taking this into account, both rotations and patch location would likely take a similar amount of time to train. The jigsaw pretext task gave an even higher accuracy, which is not surprising as, conceptually, it is a more difficult task than the other two, and has more output classes, meaning that it needs to learn to extract features very well to be able to solve the jigsaws.

The loss plot of the rotation task in the second stage (figure 4) looks very different to that of the other two tasks (figures 8 and 11). The loss of the rotation task drops significantly during the fine tuning stage, whereas the other two methods had very little to no improvement during this stage. This shows that the patch location and jigsaw tasks trained the network to extract features very well, meaning they didn't need much fine tuning. However, the rotation task did not manage to create a network that was as good at extracting features, since it gained a significant improvement during the fine tuning stage. Despite this, the rotation task was not much less accurate than the patch location task.

8 Conclusion

In the today's modern age, there is no shortage of data, however preparing it for use in machine learning techniques is very time consuming. To get accurate models, we typically need thousands of data points, each individually labelled by a human. In this report I've demonstrated how semi supervised learning can be used to train models using very little labelled data, and a large dataset without labels. The unlabelled dataset contains many images of objects that do not belong to one of the 10 classes, showing that these techniques do not require a human to sort through the unlabelled data to make sure that only the desired classes are present.

I found that training a model to do simple tasks, such as detecting the rotation applied to an image, can provide a significant improvement to the accuracy of the model, when compared to just training on labelled data. More complex tasks such as solving a jigsaw of each image allows the model to gain superior feature extraction abilities. This results in even higher accuracy's after training on the data set. Using this technique the model managed an accuracy 14.4% higher than if the network was just trained on the training dataset. These algorithms show that the large amounts of data available to us can be leveraged to improve our models where labelled data is limited or difficult to obtain.

Bibliography

[1] *Semi-supervised learning*. 2020. URL: https://en.wikipedia.org/wiki/Semi-supervised_learning.

[2] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: (2015). URL: arXiv:1512.03385v1.

[3] Andrew Y. Ng Adam Coates Honglak Lee. "An Analysis of Single Layer Networks in Unsupervised Feature Learning". In: *AISTATS* (2011). URL: http://cs.stanford.edu/~acoates/stl10.

[4] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. "Unsupervised Representation Learning by Predicting Image Rotations". In: (2018). URL: arXiv:1803.07728.

[5] Carl Doersch, Abhinav Gupta1, and Alexei A. Efros. "Unsupervised Visual Representation Learning by Context Prediction". In: (2015). URL: arXiv:1505.05192.

[6] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. "Revisiting Self-Supervised Visual Representation Learning". In: (2019). URL: arXiv:1901.09005v1.

[7] Mehdi Noroozi and Paolo Favaro. "Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles". In: (2016). URL: arXiv:1505.05192.