Self-Supervised Learning Methods on Room Type Classification

Gengxiao Li
Stanford University
Mclean, VA
gli01@stanford.edu

Satita Vittayaareekul
Stanford University
Amsterdam, The Netherlands
satita97@stanford.edu

June 9, 2023

1 Abstract

This report presents a novel approach to room type classification in a domestic setting using self-supervised learning. Our work focuses on leveraging large pre-trained models such as MAE, DINO v2 and SimCLR to perform downstream classification tasks on room types. The research change the encoder of SimCLR from CNN to ViT and compare the performance of original SimCLR model and performance after applied the pre-trained parameters of MEA and dino v2 to the room type images.

2 Introduction

As the world increasingly embraces smart home technologies and automation, the need for accurate room type classification algorithms has become paramount. These algorithms are instrumental in enhancing home automation, facilitating robot navigation, fortifying security systems, and other AI applications. Our motivation to address this challenge lies in the potential improvements these systems could bring to various domestic and commercial applications, improving lives and operational efficiency.

The problem at hand involves developing a self-supervised learning model that can accurately classify different room types in a dynamic domestic environment using visual input. Such a model should account for factors like furniture repositioning, lighting changes, and other environmental alterations affecting room appearances. The input to our algorithm is an image, and we leverage pre-trained self-supervised learning models like SimCLR, MAE, and DINO v2 to output a predicted room type, such as a living room, kitchen, bedroom, or bathroom.

Supervised learning models currently in use require extensive labeled datasets, which can be costly, time-consuming, and subject to human error. Additionally, these models struggle to generalize their learning to unseen environments. To counter these challenges, our focus is on self-supervised learning, where models learn from unlabeled data, creating useful representations for downstream tasks. We aim to utilize these representations to accurately classify room types in a dynamic environment.

3 Related Work

MAE (Masked Autoencoders)[1], our baseline model, shows a unique way of masking and reconstructing images. It develops an asymmetric encoder-decoder architecture, where the encoder operates the portion of images without mask tokens and decoder reconstructs the original image from the latent representation and mask tokens. This autoencoder is able to mask a high proportion of the input image (75%) and still accelerate the training and improve accuracy compared to many state-of-the-art models (BEiT, ViT, MoCo).

Scene Classification[2], one of our main task, is to classify a scene image into different predefined scene categories based on the content, objects, and layout

in the image provided. The current challenges in this field includes large intraclass variations (different lighting conditions, different angles of the images and different objects in the content), semantic ambiguity (many different scenes in different categories could have the same objects/texture/backgrounds), and computational efficiency. Zeng and Liao (2021) compares between different global CNN features based methods, spatially invariant features based methods, semantic features based methods, multilayer feature based methods, multiview features based methods and reveals that FTOTLM achieves the best accuracy out of other state-of-the-art models on three benchmark datasets they used.

DINO v2[3], is one of the other model we are going to use to compare with MAE baseline model. DINO v2 is a variety of pretrained visual models, the first method that uses self-supervised learning to train computer vision models. Like self-supervised learning, it does not require large amounts of labeled data, so it solves the issue of not having enough labeled data to train and evaluate. It also does not require any fine-tuning.

SimCLR[4], a simple framework for contrastive learning of visual representations which does not require specialized architectures or a memory bank. SimCLR shows the importance of data augmentation and uses random corp and resize (with random flip), color distortions, and Gaussian blur as their default settings. And reveals that stronger data augmentation is more beneficial for unsupervised contrastive learning compared to supervised learning.

ViT[5], is one of the encoder we choose to use as a comparison to CNN based encoder. It is a pure transformer applied directly to sequences of image patches without the need for convolutional layers, which remarkably proficient in image classification tasks. This paper also shows that ViT attains outstanding outcomes when comparing to state-of-the-art convolutional networks, in addition to significantly fewer computational resources in training.

4 Methods

The goal of this research is to explore different types of Self-Supervised Learning methods, and see how it works for the downstream task, which is room type classification. The methods we explored are MAE, DINO v2 and SimCLR. For MAE and DINO v2, since the training volume is relative large for these models (for example the training set of DINO v2 contains 145m images), and given limited resources and time we have, we decided not to change the model architecture on these models and directly use their open source pre-trained parameters for the room type classification. On the other hand, we modified the model architecture of SimCLR, where we changed the CNN based encoder to ViT based encoder, and trained the model from scratch. Some detailed methodology of these Self-Supervised learning models will be provided below.

The SimCLR (Simple Framework for Contrastive Learning of Visual Representations) is a self-supervised learning model developed by Google. This model uses contrastive learning, a methodology that distinguishes similar (positive) and dissimilar (negative) examples from an unlabeled dataset.

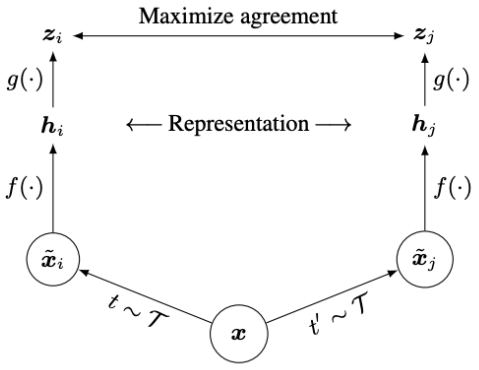

Figure 1: SimCLR[4]

The pipeline of SimCLR involves four steps: Data Augmentation: It applies two random transformations T to an input image x to obtain two correlated views x' and x'', forming a positive pair. A base

encoder network (f), often a deep Convolutional Neural Network (CNN), maps the augmented images to the feature space. These projections are then passed through a projection head (g) to obtain representations $z = g(f(x'))$ and $z' = g(f(x''))$. Then it uses the NT-Xent (Normalized Temperature-scaled Cross-Entropy) loss function. For a given positive pair, this loss aims to maximize agreement between z and z' while minimizing it among all other pairs in a mini-batch. It uses a large batch size and leverages a temperature parameter (τ) for better performance. Mathematically, the NT-Xent loss for a positive pair (i, j) is given by:

$$L(i,j) = -\log\left(\frac{\exp(\text{sim}(z_i,z_j)/\tau)}{\sum_{k\neq i}\exp(\text{sim}(z_i,z_k)/\tau)}\right)$$

where sim is the cosine similarity, and the sum in the denominator is over all negative pairs in the batch for i.

After computing the contrastive loss, backpropagation is performed to update the parameters of the encoder and the projection head. The outputs of SimCLR are representations that capture the similarity of augmented views, making them useful for various downstream tasks. While it does not provide explicit labels, it can distinguish between similar and dissimilar instances, giving rise to meaningful representations for a wide range of visual recognition tasks. SimCLR thus offers a powerful mechanism to leverage unlabeled data in computer vision.

In this project we change the CNN based encoder to ViT based encoder in SimCLR. And for Vision Transformer (ViT), it is a model for computer vision tasks introduced by Google Research. It is an application of the Transformer model to perform image classification tasks. ViT takes as input a sequence of image patches and treats them as if they were tokens in a text input. The process begins by partitioning an input image into small patches, flattening them into 1D arrays, and then linearly transforming each into a "patch embedding". To account for lost positional information during flattening, positional embeddings are added to the patch embeddings. The resulting sequence of patch embeddings is then passed through a standard Transformer encoder comprised of multiple layers of self-attention and feed-forward neural networks. The final output, corresponding to the embedding of the first input token, is used for classification through a feed-forward network, predicting the image's class. ViT, though requiring more data and computational power, has exhibited impressive results on image classification tasks when pre-trained on large datasets.

When changed the model architecture of SimCLR, we used the base code from CS231n assignment 3, and modified from there. There are two major changes:

1. Adding another Mvit class in cs231n/simclr/model.py file to create a ViT encoder inside SimCLR instead of CNN.

```
29 class Mvit(nn.Module):
30     def __init__(self, feature_dim=128):
31         super(Mvit, self).__init__()
32
epoch: int  # encoder
34         self.f = timm.create_model('vit_base_patch16_224', pretrained=False)
35         self.f.head = nn.Identity()  # remove final classification layer
36
37         # projection head
38         self.g = nn.Sequential(nn.Linear(768, 512, bias=False), nn.BatchNorm1d(512),
39                                nn.ReLU(inplace=True), nn.Linear(512, feature_dim, bias=True))
40
41     def forward(self, x):
42         x = self.f(x)
43         feature = torch.flatten(x, start_dim=1)
44         out = self.g(feature)
45         return F.normalize(feature, dim=-1), F.normalize(out, dim=-1)
```

Figure 2: Mvit Class

2. Added additional two functions: compute_train_transform_vit and compute_test_transform_vit to resize the image to 224 x 224 in cs231n/simclr/data_utils.py since vit only takes image with 224 x 224 resolution.

Next is MAE, useful for dimensionality reduction, anomaly detection, or denoising. The network, composed of an encoder and decoder, is trained to minimize the difference between the input and the reconstructed output. A Masked Autoencoder enhances this by applying a binary mask to the input, setting a portion of the input features to zero. This encourages the network to learn a more robust latent representation that capable of handling missing or corrupted data. By measuring the difference between the original input data and the reconstructed data, the Masked Autoencoder is encouraged to learn rep-

```python
def compute_train_transform_vit(seed=123456):
    """
    This function returns a composition of data augmentations to a single training image.
    Complete the following lines. Hint: look at available functions in torchvision.transforms
    """
    random.seed(seed)
    torch.random.manual_seed(seed)
    color_jitter = transforms.ColorJitter(0.4, 0.4, 0.4, 0.1)

    train_transform = transforms.Compose([
        transforms.Resize((224,224)),
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomApply([color_jitter], p=0.8),
        transforms.RandomGrayscale(p=0.2),
        transforms.ToTensor(),
        transforms.Normalize([0.4914, 0.4822, 0.4465], [0.2023, 0.1994, 0.2010])])
    return train_transform

def compute_test_transform_vit():
    test_transform = transforms.Compose([
        transforms.Resize((224,224)),
        transforms.ToTensor(),
        transforms.Normalize([0.4914, 0.4822, 0.4465], [0.2023, 0.1994, 0.2010])])
    return test_transform
```

Figure 3: Function to Resize the Image for ViT

resentations that are insensitive to specific features masked out during training. This makes it ideal for tasks involving incomplete or noisy data and robust data representation learning.

DINO V2 is a self-supervised learning model designed by Facebook AI. The architecture consists of a teacher network and a student network. The student network learns to mimic the teacher network, which evolves over time. The primary innovation is the use of a dynamic teacher network that changes over the course of training, ensuring that the model continuously learns new representations. The

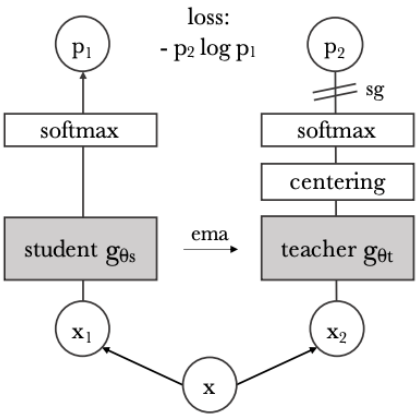

Figure 4: DINO Distillation system[6]

loss function, inspired by knowledge distillation, encourages the student network's output to align with the teacher network's output. More precisely, for a given input image, the student's outputs at multiple 'views' (transformed versions of the input) are made to match the teacher's output for a corresponding view. However, a key difference with typical distillation is that a centering operation is applied on both student's and teacher's output to prevent collapsed solutions. This centering operation subtracts the average output over a mini-batch of data.

Formally, let's denote f_s and f_t as the student's and teacher's output functions, and v as a view of an input. The loss function can be formulated as:

$$L(\theta_s) = \mathbb{E}_{v \sim V} D_{KL}(f_s(v;\theta_s)||f_t(v)) \qquad (1)$$

where θ_s represents the student's parameters, D_{KL} is the KL-divergence, and the expectation is over all views v from a set of views V. This strategy enables the model to learn rich, discriminative features even in the absence of labels.

5 Dataset and Features

The dataset we used for downstream task is the NYU-Depth V2 dataset[7]. It comprised of video sequences from a variety of indoor scenes as recorded by both the RGB and Depth cameras from the Microsoft Kinect. The resolution of the image is 640 x 480. It contains varies images of different room types. We mainly focused on the following room types: bedroom, bathroom, kitchen, living room, dining room, basement. We combined the images in the same type together. Since the raw image is in .jpg format and the corresponding depth map of each image is in .png format, we removed the .PNG files in each of the categories, and only keep the raw images. There are 2548 images in total after filtering. Next used ImageFolder to load the data and transformed it into tensors, resized it to 224 x 224, then used DataLoader to load the dataset in batches and randomly shuffled the data orders. The batch size we used is 64. Finally divided the data into 70% of training and 30% of testing, the random state is 42.

For the training process of SimCLR with ViT encoder, initially we used CIFAR10 dataset. Because we utilized the code from cs231n assignment 3, in which it used CIFAR10 to train the original SimCLR with CNN based encoder for 18 hours and saved the pre-train weights. We aim to compare the performance different between the SimCLR with CNN based encoder and SimCLR with ViT based encoder. Thus, we trained both models with the same CIFAR10 dataset. The training size for ViT based encoder of SimCLR is 50,000, and the resolution is 32 x 32. We did some data augmentation and transformations. Randomly resize and crop to 32x32. Horizontally flip the image with probability 0.5 With a probability of 0.8, apply color jitter With a probability of 0.2, and convert the image to grayscale. The normalized the R, G, B channel with their mean [0.4914, 0.4822, 0.4465] and standard deviation [0.2023, 0.1994, 0.2010]. These augmentation and transformation steps are aligned with the original SimCLR model (CNN based encoder) during the training process, except one thing needs to be changed. Since ViT only takes images with resolution 224 x 224 as its input, we need to resize the image from 32 x 32 to 224 x 224 in the beginning of the data augmentation and transformation process.

Next we brought the imagenet dataset and incorporate it into the training process of this ViT based encoder for SimCLR. Since the performance of the modified SimCLR is poor (will share more detailed information on the results and the reason for choosing another dataset in next section). Because the huge volume of imagenet dataset, and given limited resouces we have. We decided to use the validataion set of imagenet to train our ViT encoder based SimCLR model. The size of the validation set is 50,000, and the average resolution is 469 x 387. The augmentation and transformation processes are exactly the same as above, except the in the normalize step the mean [0.485, 0.456, 0.406] and std [0.229, 0.224, 0.225] for R, G, B channels changed.

6 Experiments, Results and Discussion

6.1 Experiments

In general, there are two sets of experiment we worked on.

1. Training the SimCLR with changing the model architecture, with both CIFAR10 dataset and imagenet dataset.

2. Testing on 5 different pre-trained parameters on room type classification.

The 5 pre-trained parameters are from MAE, DINO v2 (these two are directly downloaded from the original models online), SimCLR (CNN based encoder) trained with CIFAR10 dataset in 18 hours, SimCLR (ViT based encoder) trained with CIFAR10 dataset for 20 hours, SimCLR (ViT based encoder) trained with val imagenet dataset for 20 hours.

First, for the training process of SimCLR with ViT based encoder, we used CIFAR10 as the training set, which contains 50,000 images. The batch size is 16, since anything above 16 will give us memory overflow error. The optimizer is adam and the learning rate is 0.001, which are the same as the setups in the assignment. Since we utilized the code from the assignment as our base code to modify with, we did not change these settings. We used A100 GPU to train the modified version of SimCLR for 10 hours of 40 epochs, and then used T4 GPU to continue train it for another 10 hours of 10 epochs in Colab, and saved the trained weights.

Then, in order to test our hypothesis of the difference on comparing the test results of CNN based encoder and ViT based encoder of SimCLR. We ran another training process on the validation set of imagenet with the ViT based encoder SimCLR model. The validation set of imagenet contains 50,000 images. We kept the setups the same as training on CIFAR10 dataset. We used T4 GPUs training for 20 hours of 20 epochs, and saved the trained weights. The only difference is to add another ImageNetPair class to process the val ImageNet dataset.

```
# SimCLR ViT CIFAR10
feature_dim = 128
temperature = 0.5
k = 200
batch_size = 16
epochs = 50
temperature = 0.5
percentage = 0.5

# Prepare the data.
train_transform = compute_train_transform_vit()
train_data = CIFAR10Pair(root='data', train=True, transform=train_transform, download=True)
train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True, num_workers=16)
test_transform = compute_test_transform_vit()
memory_data = CIFAR10Pair(root='data', train=True, transform=test_transform, download=True)
memory_loader = DataLoader(memory_data, batch_size=batch_size, shuffle=False, num_workers=16, pin_memory=True)
test_data = CIFAR10Pair(root='data', train=False, transform=test_transform, download=True)
test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False, num_workers=16, pin_memory=True)

# Set up the model and optimizer config.
model = MyVit(feature_dim)
model = model.to(device)
flops, params = profile(model, inputs=(torch.randn(1, 3, 224, 224).to(device),))
flops, params = clever_format([flops, params])
print('# Model Params: {} FLOPs: {}'.format(params, flops))
optimizer = optim.Adam(model.parameters(), lr=1e-3, weight_decay=1e-6)
c = len(memory_data.classes)

# Training loop.
results = {'train_loss': [], 'test_acc@1': [], 'test_acc@5': []} #<< -- output

if not os.path.exists('results'):
    os.mkdir('results')
best_acc = 0.0
for epoch in range(1, epochs + 1):
    train_loss = train(model, train_loader, optimizer, epoch, epochs, batch_size=batch_size, temperature=temperature, device=device)
    results['train_loss'].append(train_loss)
    test_acc_1, test_acc_5 = test(model, memory_loader, test_loader, epoch, epochs, c, k=k, temperature=temperature, device=device)
    results['test_acc@1'].append(test_acc_1)
    results['test_acc@5'].append(test_acc_5)

    # Save statistics.
    if test_acc_1 > best_acc:
        best_acc = test_acc_1
        torch.save(model.state_dict(), './pretrained_model/trained_simclr_vit_model.pth')
```

Figure 5: Train Process for SimCLR(ViT) with CIFAR10

```
# SimCLR ViT Val Imagenet
feature_dim = 128
temperature = 0.5
k = 200
batch_size = 16
epochs = 20
temperature = 0.5
percentage = 0.5

# Prepare the data.
train_transform = compute_train_transform_vit()
train_data = ImageNetPair(root='data', split='val', transform=train_transform)
print(len(train_data))
train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True, num_workers=16)

# Set up the model and optimizer config.
model = MyVit(feature_dim)
model = model.to(device)
flops, params = profile(model, inputs=(torch.randn(1, 3, 224, 224).to(device),))
flops, params = clever_format([flops, params])
print('# Model Params: {} FLOPs: {}'.format(params, flops))
optimizer = optim.Adam(model.parameters(), lr=1e-3, weight_decay=1e-6)


# Training loop.
results = {'train_loss': [], 'test_acc@1': [], 'test_acc@5': []} #<< -- output

if not os.path.exists('results'):
    os.mkdir('results')
best_acc = 0.0
for epoch in range(1, epochs + 1):
    train_loss = train(model, train_loader, optimizer, epoch, epochs, batch_size=batch_size, temperature=temperature, device=device)
    results['train_loss'].append(train_loss)
    print("train loss: ")
    print(train_loss)

    # Save statistics.
    if test_acc_1 > best_acc:
        best_acc = test_acc_1
        torch.save(model.state_dict(), './pretrained_model/trained_simclr_vit_model2.pth')
```

Figure 6: Train Process for SimCLR(ViT) with Val Imagenet

Next, we ran the tests on room type classification of MAE, DINO v2, SimCLR, SimCLR(ViT, CIFAR10), SimCLR(ViT, Imagenet) based on their pre-trained weights or saved trained weights. The steps for these models to test on room type classification are the same. First feed all the room type images into the pre-trained weights, and then obtained the embeddings for each of the images. Next, we used a 5-fold cross-validation with KNN classifier to apply to the

data_utils.py

```
class ImageNetPair(ImageNet):
    """ImageNet Dataset.
    """

    def __init__(self, root, split, transform=None):
        root = self.root = os.path.expanduser(root)
        self.split = verify_str_arg(split, "split", ("train", "val"))
#         super().__init__(self.split_folder, **kwargs)
#         kwargs.items()
        self.root = root

        self.samples = []
        self.targets = []
        self.transform = transform
        self.syn_to_class = {}
        with open(os.path.join(root, "imagenet_class_index.json"), "rb") as f:
                    json_file = json.load(f)
                    for class_id, v in json_file.items():
                        self.syn_to_class[v[0]] = int(class_id)

        with open(os.path.join(root, "ILSVRC2012_val_labels.json"), "rb") as f:
                    self.val_to_syn = json.load(f)

        samples_dir = os.path.join(root, "validation")
        for entry in os.listdir(samples_dir):
            if split == "train":
                syn_id = entry
                target = self.syn_to_class[syn_id]
                syn_folder = os.path.join(samples_dir, syn_id)
                for sample in os.listdir(syn_folder):
                    sample_path = os.path.join(syn_folder, sample)
                    self.samples.append(sample_path)
                    self.targets.append(target)
            elif split == "val":
#                 syn_id = self.val_to_syn[entry]
#                 target = self.syn_to_class[syn_id]
                target = entry
                sample_path = os.path.join(samples_dir, entry)
                self.samples.append(sample_path)
                self.targets.append(target)

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, index):
        x_i = None
        x_j = None
```

Figure 7: ImageNetPair

embeddings of these images to compute the average accuracy score, average F1-score, average precision score and average recall score after 5 trails. As for the pre-trained weights, MAE and DINO v2 models are directly downloaded online. For MAE, the weight we chose is "facebook/vit-mae-huge" and there are 630m parameters. For DINO v2, the weight we chose is 'dinov2_vitl14' and it contains 304m parameters. For SimCLR(CNN), we used the trained weights from the assignment, and it contains 24.3m parameters. For SimCLR(ViT, CIFAR10), we used the weights trained by CIFAR10, it contains 86.3m parameters. For SimCLR(ViT, Imagenet), we used the weights trained by validation set of Imagenet, which contains the same 86.3m parameters.

6.2 Results

There are 4 metrics used to assess our classification performance.

Accuracy: This is the simplest classification metric. It is the number of correct predictions made divided by the total number of predictions made. It's useful when the target classes are well balanced.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Precision: Precision is the number of true positive results divided by the number of all positive results (including those not correctly identified). It measures the proportion of actual positives was correctly classified.

$$Precision = \frac{TP}{TP+FP}$$

Recall (or Sensitivity): Recall is the number of true positive results divided by the number of all samples that should have been identified as positive. It measures the ability of the classifier to find all the positive instances.

$$Recall = \frac{TP}{TP+FN}$$

F1 Score: F1 Score is the harmonic mean of Precision and Recall and tries to balance both. It's useful when the class distribution is unbalanced.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$$

In above formulas: TP is True Positives: the model correctly predicted the positive class. TN is True Negatives: the model correctly predicted the negative class. FP is False Positives: the model incorrectly predicted the positive class. FN is False Negatives: the model incorrectly predicted the negative class.

The overall results shown in Table 1.

6.3 Discussion

As the result shown in Table 1, DINO v2 with its pretrained parameters gives us the best performance. Since DINO v2 uses ViT as its encoder and also utilized this student-teacher distillation technique combined with a large, curated, and diverse dataset to train the models can really provide us the high performance pre-trained weights for downstream tasks. The MAE model achieved moderate results, with an accuracy of 0.66 and F1-score of 0.65. This might be due to the fact that this self-supervised learning method leverages an autoencoder for representation learning, which may not be as effective as other methods for complex image-based tasks such as room type classification. Although not as perfection as DINO v2, SimCLR also achieved high scores. This could due to its contrastive learning technique, which is effective at learning rich feature representations from unlabelled data. Still, the high scores warrant a check for overfitting. SimCLR with ViT encoder trained on CIFAR10 had the lowest scores, our hypothesis is that since we trained the ViT based encoder on top of CIFAR10 dataset, it only has 32 x 32 resolution. However, the input for ViT requires larger images, which is 224 x 224 resolution images. Resizing the images from 32 x 32 to 224 x 224 can lead to losing valuable information, When you resize such small images to a larger size, it may introduce blurriness or other distortions that affect the image's key features, leading to the lower performance you observed. Another reason is transformers can be powerful, but they often require large amounts of data and can be more challenging to train. The reduced performance could be due to the limited amount of data (50,000 images of CIFAR10) or insufficient training time (50 epochs might not be enough for the ViT to fully learn).

In order to test our hypothesis for low performance on ViT based encoder of SimCLR which trained on CIFAR10 dataset. We use the same model architecture to train on the validation set of ImageNet. As for the results from training the same ViT-based SimCLR model on the ImageNet validation set, the increase in accuracy and F1-score supports our hypothesis. Given that the ImageNet dataset's image size is naturally compatible with ViT, it makes sense to see better performance with this dataset. Although the improvements seem marginal, they do indicate that ViT-based models perform better on larger, high-resolution images, which aligns with their design intention.

However, one important consideration is that the Im-

	Accuracy	F1 Score	Prec Score	R Score
MAE	0.656	0.653	0.661	0.656
DINO v2	1.00	1.00	1.00	1.00
SimCLR	0.98	0.98	0.99	0.97
SimCLR(ViT CIFAR10)	0.52	0.49	0.49	0.49
SimCLR(ViT ImageNet)	0.55	0.54	0.53	0.54

Table 1: The Results for Room Type Classification

ageNet validation set is still relatively complex and diverse compared to the CIFAR10 dataset, which could present additional challenge to the model's learning process. Training ViT models effectively often requires large-scale datasets due to the model's capacity for learning from large amounts of data. Despite this, our experiment provided useful insights into how ViT performs under different data conditions.

These results provide valuable insights into how different self-supervised learning methods and encoders perform on various datasets. More importantly, they highlight the importance of considering dataset characteristics, such as image size and complexity, when choosing a model or encoder for your task.

7 Conclusion and Future Work

In this work, we did a comparison on 5 different pretrained weights (MEA, DINO v2, SimCLR with CIFAR10 based encoder, SimCLR with ViT based encoder that trained with CIFAR10 dataset, and SimCLR with ViT based encoder trained on ImageNet validation dataset) on room type classification. We carefully study the difference of each model and reasons for why each of the above pre-trained weight leads to the shown results.

For future works, we would like to try out masked autoencoders on SimCLR and train it on both CIFAR10 and ImageNet validation datasets and compare with the results we obtained above. Moreover, we would also want to do segmentations on top of room classification, which is to divide a room into different parts, including ceilings, floors, and walls. This will help us fulfill the initial idea of holding an automating house appraisal score calculations, which is the score shown on appraisal report for the bank to determine how much home financing they would provide to the new house owner.

8 Contributions Acknowledgements

All authors contributed equally to this project. Gengxiao Li collected and preprocessed the room type image data, modified the SimCLR encoder from CNN to VIT, Satita Vittayaareekul set up the ImageNet dataset and created ImageNetPair class to process the data. Both of us worked on training the models and analyzing the results. All papers were written jointly.

References

[1] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022. 1

[2] Delu Zeng, Minyu Liao, Mohammad Tavakolian, Yulan Guo, Bolei Zhou, Dewen Hu, Matti Pietikäinen, and Li Liu. Deep learning for scene classification: A survey. *arXiv preprint arXiv:2101.10531*, 2021. 1

[3] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil

Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023. 2

[4] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020. 2

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. 2

[6] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers, 2021. 4

[7] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012. 4