Compressed Vision for Efficient Video Understanding

Olivia Wiles*, João Carreira, Iain Barr, Andrew Zisserman, and Mateusz Malinowski*

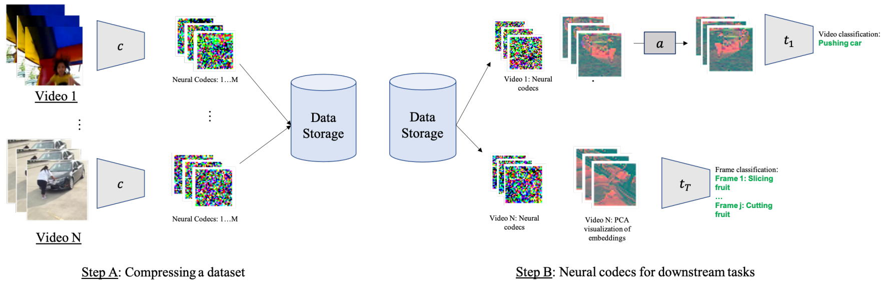
DeepMind, London, U.K.
* {oawiles,mateuszm}@deepmind.com

Abstract. Experience and reasoning occur across multiple temporal scales: milliseconds, seconds, hours or days. The vast majority of computer vision research, however, still focuses on individual images or short videos lasting only a few seconds. This is because handling longer videos require more scalable approaches even to process them. In this work, we propose a framework enabling research on hour-long videos with the same hardware that can now process second-long videos. We replace standard video compression, e.g. JPEG, with neural compression and show that we can directly feed compressed videos as inputs to regular video networks. Operating on compressed videos improves efficiency at all pipeline levels – data transfer, speed and memory – making it possible to train models faster and on much longer videos. Processing compressed signals has, however, the downside of precluding standard augmentation techniques if done naively. We address that by introducing a small network that can apply transformations to latent codes corresponding to commonly used augmentations in the original video space. We demonstrate that with our compressed vision pipeline, we can train video models more efficiently on popular benchmarks such as Kinetics600 and COIN. We also perform proof-of-concept experiments with new tasks defined over hour-long videos at standard frame rates. Processing such long videos is impossible without using compressed representation.

Keywords: Video, Long-Video, Compression, Representation

1 Introduction

Most computer vision research focuses on short time scales of two to ten seconds at 25fps (frames-per-second) because vision pipelines do not scale well beyond that point. Raw videos are enormous and must be stored compressed on a disk; after loading them from a disk, they are decompressed and placed in a device memory before using them as inputs to neural networks. In this setting, and with current hardware, training models on minute-long raw videos can take prohibitively long or take too much physical memory. Even loading such videos onto GPU or TPU might become infeasible, as it requires decompressing and transferring, often over the bandwidth-limited network infrastructure. While


c
Video 1
Neural Codecs: 1…M
Video N
Neural Codecs: 1…M
Data Storage
Data Storage
Video 1: Neural codecs
a
t_1
Video classification: Pushing car
Video N: Neural codecs
Video N: PCA visualization of embeddings
t_T
Frame classification: Frame 1: Slicing fruit … Frame j: Cutting fruit
Step A: Compressing a dataset
Step B: Neural codecs for downstream tasks

(a) (b)

Fig. 1: **The compressed vision pipeline.** Videos are first compressed using a *neural compressor* c to produce codes. These are stored on a disk and the original videos can be discarded. The neural codes are directly used to train video tasks $t_1 \dots t_T$. We can optionally augment these codes with augmented versions using an *augmentation network* a (here we show a flipping augmentation). Note that within our framework, all the computations are done in a more efficient compressed space as decompression is not required at any stage of the pipeline.

previous work has considered using classic video or image compressed codes (such as JPEG or MPEG) directly as input to their models [1,2,3,4], this generally requires specialised neural network architectures.

In this work, we propose and investigate a new, efficient and more scalable video pipeline – *compressed vision* – which preserves the ability to use most of the state-of-the-art data processing and machine learning techniques developed for videos. The pipeline, described in more detail in Section 3, has three components. First, we train a *neural compressor* to compress videos. Second, we optionally use *augmentation network* to transform the compressed space for doing augmentations. Third, we *directly* apply standard video backbone architectures on these neural codes to train and evaluate on standard video understanding tasks (thereby avoiding costly decompression of the videos). As our framework is modular, each component could be replaced with a more efficient variant.

Performing augmentations (e.g. *spatial cropping* or *flipping*) is an important component of many pipelines used to train video models but are impossible to perform directly in the compressed space. Therefore, we face the following dilemma. We either give up on augmentations or we decompress the codes and do the transformations in the pixel space. However, if we decide on the latter, we loose some benefits of the compressed space. Decompressed signals expands the space and if they are long enough they cannot fit to a GPU or TPU memory anymore. Moreover, even though a *neural compressor* yields superior quality at higher compression rates to JPEG or MPEG, it has large decoders that take even more time and space; i.e., neural decompression is slow.

To overcome the last challenge, we propose an *augmentation network* – a small neural network that acts directly on latent codes by transforming them according to some operation. In brief, the *augmentation network* for spatial cropping takes crop coordinates and a tensor of latent codes as inputs. Next, it outputs the modified latents that approximate the ones obtained by spatially cropping the video frames. Note that, unlike [5], we *learn* how to augment in the compressed space as opposed to cropping the compressed tensor. As a result, we can train an *augmentation network* for a wider variety of augmentations, such as changing the *brightness* or *saturation* or even performing *rotations*; all these are difficult or impossible by directly manipulating the tensor.

Our approach has the following benefits. First, it allows for standard video architectures to be directly applied on these neural codes, as opposed to devising new architectures, as in [3] which trains networks directly on MPEG representations. Second, as demonstrated in Section 3.2, we can apply augmentations directly on the latent codes *without* the need to decompressing them. This significantly impacts training time and saves memory. With these two properties, we can use standard video pipelines with minimal modifications and achieve competitive performance to operating on raw videos (RGB values).

In summary, we show that neural codes generalise to a variety of datasets and tasks (whole-video and frame-wise classification), and are better at compression than JPEG or MPEG. To enable augmentations in the latent space, we train and evaluate a separate network, which conditioned on transformation arguments, outputs transformed latents. We also demonstrate our framework on much longer videos. Here, we collected a large set of ego-centric videos recorded by tourists walking in different cities[1]. These videos are long and continuous, and last between 30 minutes to ten hours. One can see a few samples of our results on the website[2]. We plan to update it in the future, e.g., to include the source code.

2 Related Work

Our work is built upon the following prior work.

Operating on a compressed space. Directly using compressed representations for downstream tasks for video or image data has primarily been studied by considering standard image and video codecs such as JPEG or MPEG [1,2,3], DCT [4,6] or scattering transforms [7]. However, in general these approaches require devising novel architectures, data pipelines, or training strategies in order to handle these representations. Another strategy is to learn representations that are *invariant* to a range of transformations, as in [8]. However, this requires a-priori knowledge of the downstream tasks to be invariant to. In our case, we do not have this knowledge, as we want to have the same representation to be used for a variety of potentially unknown tasks.

Discrete representations for compression. Some modalities such as language are inherently discrete and naturally benefit from neural discrete representations [9,10]. However, the same representation has become increasingly more common in generative modelling of images [11,12] and videos [13,14]. In our work,

[1] The reader can get a feel for the dataset here `https://youtu.be/MIzp8Wrj44s?t=131`.

[2] Project website: `https://sites.google.com/view/compressed-vision`

we demonstrate a different use of vector quantization: to compress videos. Thus, we can directly train classifiers on the resulting compressed space, without the need for decompression, making one-hour long training and inference feasible.
Augmentations using latents. In the image domain, it has been demonstrated that augmentations in an embedded space learned via a GAN [15,16,17] or an encoder-decoder model [18] can be used to improve classification performance. In the video setting, [5] have applied augmentations directly in a latent space, but only considered cropping of the tensor – an operation that can readily be designed. Instead, we propose a learnable approach to approximate augmentations in a latent space, so that we can generalize the approach to arbitrary augmentations.
Video understanding. Understanding long videos encompasses many important sub-problems including action or event understanding and reasoning over longer spatio-temporal blocks. It is also computationally demanding. On these tasks, transformer-like architectures [19,20,21,22,23,24] could potentially compete with spatio-temporal convolutional networks (3D CNNs) [25,26,27,28,29] that reason about time and space locally. However, videos in current datasets are often too short [30,31,32,33,34] and thus these data hungry architectures are not so useful. This work demonstrates how neural compression can be used to scale architectures to operate on much longer sequences.
Long sequence processing. Long-Range Arena [35] was designed to challenge the ability of transformer-like architectures to model long-term dependencies and their efficiency. These problems are well handled directly by efficient transformers [36,12,37,38]. Walking Tours also sets up a benchmark for efficient and long-term video understanding by probing networks on their ability to handle long sequences where input signals are high-dimensional.

3 Compressed Vision

Here, we introduce our *compressed vision* pipeline. Next, we discuss each component of the pipeline.

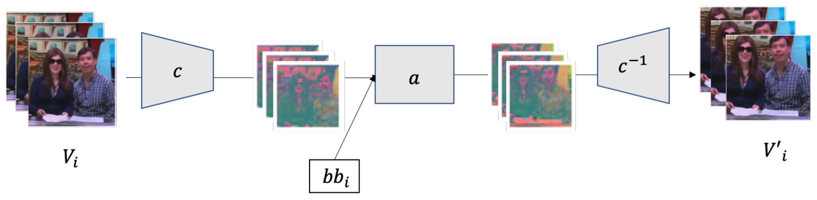


Fig. 2: **Augmentation Network.** We train a network a that, conditioned on the bounding box coordinates of the desired spatial crops, performs that spatial crop directly on the latent codes (these embeddings are visualised using PCA). As shown here, after decompression, the resulted video corresponds to spatially-cropped original video. Even though cropping is our running example, our methodology extends to potentially arbitrary augmentations.

3.1 Our Pipeline

The typical setup used for training video networks is motivated by that videos are stored in a compressed form. To train neural networks, these videos need to be loaded into a memory and decompressed; taking up space and time.

Instead, it would be preferable to operate directly on the neural codes. Other methods that have investigated this setup used the MPEG encoding directly [3]. However, this requires devising specific architectures and pipelines to handle compressions in that form. As a result, a significant amount of work and progress in the last ten years to apply neural networks to video data cannot be directly applied to these representations. Instead, our pipeline creates compressed tensors that are amenable to being loaded efficiently and passed directly as input to standard video pipelines with minimal, if any, changes in hyper-parameters.

An overview of our pipeline is given in Figure 1. It operates in three stages. First, we have a *neural compressor* c that is used to compress videos a-priori and store them on a disk or other data carriers (Figure 1a). After learning this network, there is no further need to touch the original videos and in fact they can be removed to free-up space. Second, for any downstream task, we have a downstream network t_i that is applied to these neural codes to solve that task (Figure 1b). These tasks may be varied (e.g. frame prediction, reconstruction, classification, etc.) but they all use the same neural codec. Additionally, as we use spatial tensors, many standard architectures can be used while only changing hyper-parameters of the model. Finally, we also learn an *augmentation network* a that (Figure 2), conditioned on a neural code R, can transform it into a new representation $R' = c(X)$. When decompressed using a decoder c^{-1}, R' approximates an augmentation transformation A, e.g. a spatial crop, acting on the corresponding rgb-valued video X. That is, we are aiming at $c^{-1}(a(c(X))) \approx A(X)$.

3.2 Learning Neural Codecs

To learn c, we build on a standard VQ-VAE encoder-decoder model [10]. The VQ-VAE model uses an encoder to map images into a spatial tensor. Next, each vector in the spatial tensor is compared with a sequence of embeddings using nearest neighbours. Indices that correspond to the chosen embeddings are stored as codes. These codes represent the input images, which can now be discarded: we need only store codes for each image and the fixed-length sequence of embeddings, which we call the codebook. To decode, the codes are mapped to the corresponding embeddings from the codebook, and passed to the decoder. To extend such a representation to videos simply requires modifying the encoding and decoding networks. Instead of using 2D spatial convolutions, we use 3D convolutions to obtain a 3D tensor of codes. We can additionally use additional codebooks for improved performance at the cost of using more memory.

Discussion. There are three factors controlling compression rate. First, the size $T_T \times T_H \times T_W$ of the tensor learned by the encoder controls its spatio-temporal size. Second, the number of codebooks T_C controls the number of codes stored at each location in the tensor. Finally, the number of codes K in the codebook controls the number bits required to store a code. An RGB-video, $I_T \times I_H \times I_W \times 3$, gives a compression rate: $c_r = \frac{I_T I_H I_W * 3 * \log_2 256}{T_T T_H T_W T_C \log_2 K}$. Although we focus on compressing space, we provide promising time-compression results in the supplementary.

Training general representations. We train the discrete codes using an auto-encoding task and l_1 reconstruction loss. The codes should encode a neural codec that approximates the original video while removing redundancy. As a result, we will be able to use it for a variety of (potentially unknown) downstream tasks (e.g. classification). Moreover this representation should be useful for finding augmentations in that space as different videos (including augmented ones) will map to a separate representation. Note that the intuition here is *not* that we are learning an invariant representation as in [8], which would impede the ability to learn augmentations in the representation space.

Encoder/decoder architecture. First, we create spatio-temporal patches of the videos. Next, we use strided 3D CNNs with inverted ResNet blocks [39] to obtain a spatial tensor of vectors. We independently quantize the resulting vectors. This gives intermediate representations that are reversed using strided transposed convolutions for decoding. Further details about the architecture and hyperparameters are given in the supplementary material.

3.3 Video Tasks with Neural Codes

Our final aim is to use the neural codes as the input for arbitrary video tasks. As our neural codes have a spatio-temporal structure, we can train standard modern architectures, such as S3D [29], directly on this representation for the downstream task. During training, we sample codes that correspond to videos, and obtain the corresponding embeddings using the codes as indices. We input these tensors to our downstream model, e.g. S3D [29]. To maintain the same spatial resolution as the original S3D, we modified the strides of the convolutions. For example, for a compressed tensor of size 28×28 we use a stride of 1 (versus 2 for a 224×224 image).

3.4 Augmentations in the Compressed Space

In a standard vision pipeline, being able to augment the input at train and evaluation time often leads to large boosts in performance. Here we describe how a similar procedure can be applied to our neural codes for similar gains.

The central idea is to learn to approximate augmentations but in the compressed space. We use a relatively small neural network, which we call an *augmentation network*, to learn such an approximation. Let A be an augmentation transformation on the input video X. For instance, $A(X)$ can spatially crop the input signal X. Given X and the bounding box bb describing the coordinates of the crop, we train a neural network $a(\cdot)$ to perform the equivalent transformation but in the compressed space. If c and c^{-1} denote encoder and decoder, we want to maintain the following relationship: $A_{bb}(X) = c^{-1}(a(c(X), bb))$.

Downstream tasks. As in a traditional pipeline, we can apply augmentations at both train and test time to boost performance. In our pipeline, we proceed as follows and use an *augmentation network* trained, for example, to predict spatial crops. At train time, for each compressed video clip X in a batch, we randomly select a bounding box bb and then apply the transformation $a(X, bb)$ to obtain a transformed version of the video clip X' which is the input to the downstream network. At test time, for a given compressed video clip X, we linearly interpolate N bounding box coordinates. Each of these are used to transform the video clip to create N augmented versions of each video clip. The predictions for all N clips are averaged to obtain the final prediction for that clip.

Training. We train the *augmentation network* after learning the neural codec (so there is no requirement that these steps are done on the same datasets) and keep c and c^{-1} fixed. To train, we select a given augmentation class (such as spatial cropping) which is parameterised by some set of values (such as the bounding box for spatial cropping). We then create training pairs by randomly selecting pairs: a video X and a bounding box bb. We apply the augmentation to the video to get its augmented version X'. Finally, we minimize the following reconstruction loss (we use an l_1 loss) to train the corresponding transformation network:

$$||A(X) - c^{-1}(a(c(X), bb))||_{l_1}. \tag{1}$$

We find that using an l_1 loss is sufficient for good results on downstream tasks; we do not require more complicated training pipelines that use adversarial losses. Note that we only train a and all other transformations or networks are frozen. $a(\cdot)$ can then be applied to any video to simulate the given augmentation.

Implementation. We use a multi-layer perceptron (MLP) with three hidden layers and a two-layer transformer [40] to represent $a(\cdot)$. The MLP is applied to a tensor representing an external transformation, e.g., bounding box coordinates or binary values describing when flipping. This creates a representation that we condition on. Obtained features have the same number of channels as the neural codes. We then broadcast them over the spatial and temporal dimensions of the compressed tensor and concatenate along the channel dimension. The result is passed through the transformer to obtain a transformed representation of the same dimension as the neural codes but conditioned on the external transformation. Please see the supplementary material for the precise details.

4 Experiments

Here, we evaluate our generic pipeline using compressed video representations with latent augmentations. We test three things: (1) the utility of our neural codes for achieving high quality performance on a variety of downstream tasks; (2) the memory and speed improvements using our neural codes over using the original video frames; and (3) the ability to use standard tools, such as augmentations, within our framework to recreate the generic vision pipeline.

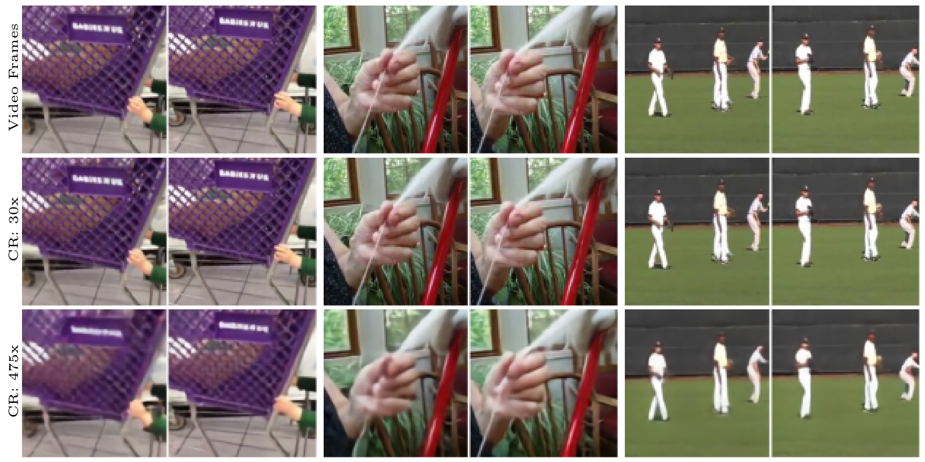


Fig. 3: **Reconstruction results at different compression levels.** At 30x compression (second row), the results are virtually indistinguishable from the original videos (first row). At 475x compression (third row), the codes lose higher frequency detail but still capture the overall structure of the scene.

We investigate this by first determining how well the compressed tensors can be used to reconstruct the original videos in Section 4.2 before discussing their utility on a variety of downstream tasks in Sections 4.3-4.6. We also investigate transfer of neural codecs if they are trained on another dataset. Next, we investigate how well our trainable transformations model augmentations in Section 4.5. Finally, we discuss the speed of a classification model when used with neural codes in Section 4.7. We show more details in the supplementary material.

4.1 Datasets

We pre-train our models on one of two datasets: Kinetics600 [30,41] and our internal dataset of recordings of tourists visiting different places. We named this dataset WalkingTours. We evaluate our models on three datasets: Kinetics600 (video classification), WalkingTours (hour long understanding), and COIN (framewise video classification) [42]. Further details are in the supplementary.

4.2 Reconstructions

We first investigate the quality of our neural codes for the task of reconstruction. Note, however, that in our work the final performance on downstream tasks is more important than the reconstruction benchmark; in the spirit of [8]. Nonetheless, we would expect these numbers to be correlated with downstream performance. We can also use reconstruction for introspection and debugging.

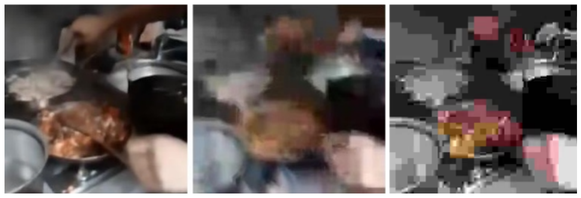


Fig. 4: **Neural vs MPEG vs JPEG codecs.** Our neural codecs (left) are better than MPEG (middle) and JPEG (right) at higher compression rates (≈180 CR).

Baselines. We compare our results to using a JPEG encoding. This is a commonly used compression scheme in video pipelines, and even though it ignores the time dimension, it offers quite good quality at low compression rates. However, unlike the frame-based JPEG, our compressors have access to neighbouring information in order to learn better representations at potentially larger compression rates. We use the OpenCV library [43] to obtain the JPEG encodings for various compression levels. We also compare the quality of our compression mechanism to MPEG, which is better than JPEG at higher compression rates.
Results. We report the performance of our compression model when reconstructing the original videos in Table 1 and visualise reconstructions for varying compression levels in Figure 3. We use standard reconstruction metrics for various levels of compression to demonstrate the reconstruction and compression tradeoffs. We can see that we can get low reconstruction error for large compression rates and that these reconstructions capture the higher level structure of the scene and are good quality. We also observe much higher degradation in quality for higher compression rates in JPEG and MPEG than our neural codec. Our visualizations in Figure 4 confirm the quantitative results in Table 1 that our codec is better than MPEG at high compression rates. This demonstrates that our representations should be informative enough to use for downstream tasks.

4.3 Video-Level Classification on Kinetics600

Next, we investigate directly using our neural codes for downstream tasks.
Baselines. We compare to the baseline of just using the original RGB frames. This is an upper bound of what we would expect to achieve. In our case, instead of inputting a tensor of frames into the downstream network for classification, we input the compressed tensors directly. To apply the S3D architecture to these compressed tensors we simply modify the stride and kernel shapes of the network (hyperparameters of the model). Note that such small modifications are common in the research on videos, e.g. when using 'space-to-depth' tricks.
Results. Results are reported in Table 2. As we can see, using 30x compression leads to a small ($\sim$ 1% drop in performance) and we can even use on the order of 256x or 475x compression with only a 5% drop in performance. In this research, we have simplified the pipeline, e.g. we use fewer iterations than commonly used, hence, our results are below SOTA using the same model.

Table 1: **Reconstruction error for codecs.** We compare our approach to using JPEG and MPEG encodings. We report three standard reconstruction metrics (PSNR, SSIM, the mean absolute error (MAE)) on Kinetics600 at different compression rates (CRs). For MAE, lower is better, for others, higher is better.

	Kinetics600		
	PSNR ↑	SSIM ↑	MAE ↓
JPEG CR~30	36.4	94.1	0.013
JPEG CR~90	25.1	70.2	0.045
JPEG CR~180	22.5	63.1	0.057
MPEG CR~30	33.2	89.6	0.034
MPEG CR~90	38.7	82.4	0.026
MPEG CR~180	23.7	67.3	0.054
CR~30	38.6	97.6	0.008
CR~236	30.8	89.8	0.019
CR~384	30.0	88.4	0.019
CR~768	29.0	85.4	0.022

We also investigate how well the neural codes transfer: if we use neural codes trained on a different dataset, say WalkingTours, how well do they transfer to Kinetics600. We find in Table 2 that representations transfer between datasets; i.e. using a representation trained on WalkingTours leads to a similar result.

Table 2: **Classification accuracy on Kinetics600.** We report Top-1 accuracy on K600 when using neural codes trained one either K600 or WalkingTours. We experiment with different levels of compression (different compression rates (CRs)). CR~1 denotes original RGB frames (without compression).

Evaluated on K600			
Trained on K600		Trained on WalkingTours	
CR	Top-1 ↑	CR	Top-1 ↑
CR~1	73.1	CR~1	73.1
CR~30	72.2	CR~30	71.3
CR~475	68.2	CR~256	68.4

4.4 Frame-level Classification on COIN

We next investigate the performance of our neural codes on a different task: framewise prediction. Unlike in the video classification task, this requires being

able to predict localised information. A representation that throws away too much information would struggle on this task. Here we investigate the utility of our neural codes on this task.
Setup. We use the same RGB baseline and setup as in the Kinetics600 case (Section 4.3). As we found in Section 4.3 that training the neural codes on a different dataset led to no loss of performance, we use our neural codes trained on either Kinetics600 or WalkingTours when training the downstream model.
Results. We report per frame accuracy in Table 3 for both datasets and different compression rates. We compute means and standard deviations over the test set. We find that, surprisingly there is virtually no loss of performance at a 30x compression rate and only minimal loss of performance at higher compression rates (e.g. 256x, 475x). Indeed, at 30x compression our model trained on Kinetics600 is even performing better than using the original RGB frames. The choice of training dataset for a neural codec has minimal impact on the downstream performance.

Table 3: **Downstream classification accuracy on COIN.** We report Top-1 and Top-5 accuracy on COIN when using neural compression trained on either K600 or WalkingTours. We experiment with different levels of compression (different compression rates (CRs)). CR~1 denotes original RGB frames.

Evaluated on COIN					
Trained on K600			Trained on WalkingTours		
CR	Top-1 ↑	Top-5 ↑	CR	Top-1 ↑	Top-5 ↑
CR~1	44.3 ± 0.3	71.7 ± 0.1	CR~1	44.3 ± 0.3	71.7 ± 0.1
CR~30	45.5 ± 0.4	73.2 ± 3.7	CR~30	44.6 ± 0.3	71.9 ± 0.3
CR~475	41.7 ± 0.5	65.6 ± 0.5	CR~256	42.7 ± 0.3	67.1 ± 0.3

4.5 Augmentations in the Compressed Space

We also investigate whether we can harness standard techniques in a vision pipeline to push performance further. A standard technique when achieving high quality results is to use spatial and temporal cropping at both training and evaluation time. While our previous results assume that we can simply save various augmented versions of the dataset, that is not feasible for larger datasets.
Setup. Here we investigate whether we can use an *augmentation network* to augment neural codes, as described in Section 3.4. We train the augmentation network on Kinetics600, and use it to augment neural codes when training a downstream model. In our experiments, we use the compression model trained on WalkingTours with 30x compression rate. We focus on spatial cropping and flipping – standard augmentations used in video tasks to improve performance.

For spatial cropping, we proceed as follows. We take a video of size 256×256 and randomly select a 224×224 crop. This defines our bounding box which we

Table 4: **Using our learnt network for augmentation.** We report Top-1 accuracy on K600. We experiment with different numbers of spatial crops at resolution 224×224 and with flipping at resolution 256×256. When using spatial crops, all models are trained using augmented spatial crops.

		Num of temporal clips			
	Crop Size	1	2	4	8
224 central crop	224	60.6	62.1	67.8	69.6
224 NN Crops (4 spatial crops) [5]	224	60.7	63.0	68.1	69.0
Ours (2 spatial crops)	224	61.6	64.1	69.1	**70.1**
Ours (3 spatial crops)	224	61.3	63.9	68.9	69.6
Ours (4 spatial crops)	224	**61.9**	**64.4**	**69.3**	69.6
256 central crop	256	60.8	62.4	68.2	68.9
Ours (with flipping at train)	256	61.7	64.4	68.5	70.0
Ours (with flipping at train and eval)	256	**62.9**	**65.2**	**69.0**	**70.2**

use to learn the *augmentation network*. At train time, we randomly select a crop. At evaluation time, we pool over linearly spaced crops to obtain the final logits. To investigate the utility of our approach, we investigate how we can improve performance at test time with additional, learned spatial crops (after training the downstream model with augmented crops).

For flipping, we take a video of size 256×256. At train time we either flip or not. At evaluation time, we pool over the original clip and its flipped version. We evaluate whether flipping (at train, eval or both) improves performance.

Baselines. For spatial cropping, we compare to two baselines. First, we consider simply using 224×224 central crops. Second, we crop directly the compressed tensor by using nearest neighbours sampling according to the bounding box resized to that spatial resolution; this baseline is similar to the approach in [5]. For flipping, we compare performance to using a 256×256 central crop.

Results. We visualise our learned augmentations in Figure 5 for spatial cropping and flipping. As can be seen, these augmentations closely match the ground truth transformation. This observation is quantitatively confirmed by the SSIM scores (0.96) between the results obtained by directly cropping in the RGB-space and using our learned *augmentation network*. As a reference the same score between unchanged videos and doing crops in the RGB-space yields only 0.44 SSIM score, showing that the ground-truth and learned crops are indeed highly correlated. We draw similar conclusions about the other two augmentations. Regarding downstream tasks, we report results in Table 4 for Kinetics600. As we can see, learning to approximate augmentations improves over all baselines, demonstrating that our pipeline can leverage standard data augmentation techniques to push performance on downstream tasks. An additional benefit of our learned approach is that we can model transformations that cannot be defined by a

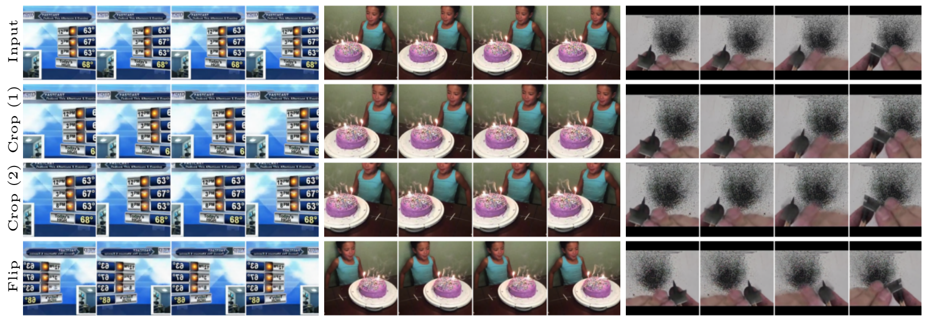


Fig. 5: **Learned transformations for augmentations.** The top row shows the original frames for three videos; the next two rows show these frames after applying our *augmentation network* for spatial cropping with two different bounding box inputs; and the bottom row shows them after applying our *augmentation network* for flipping. These results are obtained by applying our learned *augmentation network* to the corresponding compressed codes and decompressing.

simple transformation over embeddings, e.g. flipping or changing the brightness. The appendix has more such examples (e.g. saturation and rotation).

Note that the setup for these results differ from those in Table 2. In Table 2, we augment the videos before creating codes by using random spatial cropping. Here we *only* use the learned augmentation at train and test time to augment the neural codes with transformed versions. Using learned augmentations leads to a 2% drop in performance while improving over the baselines. It is because we investigate augmentations after applying central crops; effectively reducing the space of possible combinations. This is mostly an engineering limitation as the existing pipelines do cropping before assembling data into batches.

Finally, we also show that we can train our *augmentation network* to do other transformations that are unnecessary for the classification tasks, but could potentially be useful for other problems. These results also show how universal our methodology is. Figures 6 and 8 in the supplementary material show the brightness transformation. Figure 9 shows two other challenging transformations: rotations and changes in saturation. As we can see all these transformations are successfully learned. Note that all these transformations are conducted in the latent space for convenience and we decode them afterwards for visualisation.

4.6 Long-Term Video Predictions

Here, we discuss our results on long-term video prediction using WalkingTours as the dataset. Towards this goal we have created a task *past-future* that does not require supervision. *Past-future* provides a query to the network and asks the question if the short video clip (5 seconds) has been observed by the network at some point in the past. Sampled clips always come from the same video but have different time stamps. We do the inference, at train and test times, in the causal setting, where the network only sees inputs seen in the past and has no access to the future frames. All the past frames form a compressed memory and are directly accessible to the network at the query time.

Our downstream model differs from the models used above as it requires access to the memory. For that we use a transformer architecture that can effectively query the memory. We also experiment with different memories. Non-parametric *slot* refers to directly keeping all the past frames in the memory (but in the compressed form). We also use *LSTM* as the memory; it transforms a variable-length inputs into a fixed-length vector representation. Finally, *none* denotes no memory. We provide a more detailed explanation of the architecture and the task in the supplementary material. We have conducted experiments on 30 minutes long videos (training and inference). Our results give 99.6% for *slot*, 78.2% for *LSTM* and 52.9% for *none* (which is equivalent to random chance). Using one-hour long videos, with our best configuration, leads to a drop in performance of 66%, demonstrating there are still gains to be made for long video understanding.

4.7 Speed requirements

Here, we investigate whether we can train architectures faster; as our representations are smaller spatially and in total memory size. To perform this comparison we run the forward pass 100 times and report the mean and standard deviation of the time it takes on a Tesla V100. We compare the speed when using a compression rate of 256 versus no compression rate. Using compression rate of 256 leads to a forward pass that is on average 0.052 ± 0.002 seconds versus 0.089 ± 0.001 when using no compression. Thus, we can achieve about 2x speed-up with minimal loss of accuracy. The supplementary material has more comprehensive study.

5 Conclusions

This work recreates a standard video pipeline using a neural codec. After learning our neural codes, we can (1) use competitive, modern architectures and (2) apply data augmentation directly on the neural codes. The benefits of using our compressed setup is that we can (1) reduce memory requirements; (2) improve training speed; and (3) generalise to minute or even hour long videos. In order to investigate the feasibility of our approach on hour long videos, we introduced a dataset and task to explore this setting.

Our work is the first, to our knowledge, to demonstrate how to use neural codes for a set of, potentially unknown, downstream tasks by leveraging standard video pipelines, including data augmentation in the form of learned transformations on the compressed space. There are many avenues of future work to improve video understanding at scale. Some directions include improving the compression, scaling to multi-day videos, and collating tasks and datasets for such long horizon timescales.

Acknowledgements We thank Sander Dieleman for feedback on the paper. We thank Meghana Thotakuri for her help with open sourcing this work.

References

1. Gueguen, L., Sergeev, A., Kadlec, B., Liu, R., Yosinski, J.: Faster neural networks straight from jpeg. In: Advances in Neural Information Processing Systems (NeurIPS). (2018)
2. Ehrlich, M., Davis, L.S.: Deep residual learning in the jpeg transform domain. In: Proceedings of International Conference on Computer Vision (ICCV). (2019)
3. Wu, C.Y., Zaheer, M., Hu, H., Manmatha, R., Smola, A.J., Krähenbühl, P.: Compressed video action recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR). (2018) 6026–6035
4. Xu, K., Qin, M., Sun, F., Wang, Y., Chen, Y.K., Ren, F.: Learning in the frequency domain. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR). (2020)
5. Patrick, M., Huang, P.Y., Misra, I., Metze, F., Vedaldi, A., Asano, Y.M., Henriques, J.F.: Space-time crop & attend: Improving cross-modal video representation learning. In: International Conference on Computer Vision (ICCV). (2021)
6. Nash, C., Carreira, J., Walker, J., Barr, I., Jaegle, A., Malinowski, M., Battaglia, P.: Transframer: Arbitrary frame prediction with generative models. arXiv preprint arXiv:2203.09494 (2022)
7. Oyallon, E., Belilovsky, E., Zagoruyko, S., Valko, M.: Compressing the input for cnns with the first-order scattering transform. In: Proceedings of the European Conference on Computer Vision (ECCV). (2018)
8. Dubois, Y., Bloem-Reddy, B., Ullrich, K., Maddison, C.J.: Lossy compression for lossless prediction. In: Advances in neural information processing systems (NeurIPS). (2021)
9. Mnih, A., Gregor, K.: Neural variational inference and learning in belief networks. In: International Conference on Machine Learning, PMLR (2014) 1791–1799
10. Oord, A.v.d., Vinyals, O., Kavukcuoglu, K.: Neural discrete representation learning. arXiv preprint arXiv:1711.00937 (2017)
11. Esser, P., Rombach, R., Ommer, B.: Taming transformers for high-resolution image synthesis. arXiv preprint arXiv:2012.09841 (2020)
12. Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., Sutskever, I.: Zero-shot text-to-image generation. arXiv preprint arXiv:2102.12092 (2021)
13. Walker, J., Razavi, A., Oord, A.v.d.: Predicting video with vqvae. arXiv preprint arXiv:2103.01950 (2021)
14. Yan, W., Zhang, Y., Abbeel, P., Srinivas, A.: Videogpt: Video generation using vq-vae and transformers. arXiv preprint arXiv:2104.10157 (2021)
15. Chai, L., Zhu, J.Y., Shechtman, E., Isola, P., Zhang, R.: Ensembling with deep generative views. In: Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR). (2021)
16. Jahanian, A., Chai, L., Isola, P.: On the "steerability" of generative adversarial networks. In: International Conference on Learning Representations (ICLR). (2020)
17. Härkönen, E., Hertzmann, A., Lehtinen, J., Paris, S.: Ganspace: Discovering interpretable gan controls. Advances in Neural Information Processing Systems (NeurIPS) (2020)
18. DeVries, T., Taylor, G.W.: Dataset augmentation in feature space. arXiv preprint arXiv:1702.05538 (2017)
19. Bello, I., Zoph, B., Vaswani, A., Shlens, J., Le, Q.V.: Attention augmented convolutional networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. (2019) 3286–3295

20. Bertasius, G., Wang, H., Torresani, L.: Is space-time attention all you need for video understanding? arXiv preprint arXiv:2102.05095 (2021)
21. Fan, H., Xiong, B., Mangalam, K., Li, Y., Yan, Z., Malik, J., Feichtenhofer, C.: Multiscale vision transformers. arXiv preprint arXiv:2104.11227 (2021)
22. Huang, Z., Wang, X., Huang, L., Huang, C., Wei, Y., Liu, W.: Ccnet: Criss-cross attention for semantic segmentation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. (2019) 603–612
23. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems (NeurIPS). (2017)
24. Wang, X., Girshick, R., Gupta, A., He, K.: Non-local neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2018) 7794–7803
25. Carreira, J., Zisserman, A.: Quo vadis, action recognition? a new model and the kinetics dataset. In: Conference on Computer Vision and Pattern Recognition (CVPR). (2017)
26. Feichtenhofer, C., Fan, H., Malik, J., He, K.: Slowfast networks for video recognition. In: Proceedings of the IEEE International Conference on Computer Vision. (2019) 6202–6211
27. Stroud, J., Ross, D., Sun, C., Deng, J., Sukthankar, R.: D3d: Distilled 3d networks for video action recognition. In: The IEEE Winter Conference on Applications of Computer Vision. (2020) 625–634
28. Tran, D., Bourdev, L., Fergus, R., Torresani, L., Paluri, M.: Learning spatiotemporal features with 3d convolutional networks. In: Proceedings of the IEEE international conference on computer vision. (2015) 4489–4497
29. Xie, S., Sun, C., Huang, J., Tu, Z., Murphy, K.: Rethinking spatiotemporal feature learning for video understanding. arXiv preprint arXiv:1712.04851 **1** (2017) 5
30. Kay, W., Carreira, J., Simonyan, K., Zhang, B., Hillier, C., Vijayanarasimhan, S., Viola, F., Green, T., Back, T., Natsev, P., et al.: The kinetics human action video dataset. arXiv preprint arXiv:1705.06950 (2017)
31. Gu, C., Sun, C., Ross, D.A., Vondrick, C., Pantofaru, C., Li, Y., Vijayanarasimhan, S., Toderici, G., Ricco, S., Sukthankar, R., et al.: Ava: A video dataset of spatio-temporally localized atomic visual actions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2018) 6047–6056
32. Kuehne, H., Jhuang, H., Garrote, E., Poggio, T., Serre, T.: Hmdb: a large video database for human motion recognition. In: International Conference on Computer Vision (ICCV). (2011)
33. Sigurdsson, G.A., Gupta, A., Schmid, C., Farhadi, A., Alahari, K.: Charades-ego: A large-scale dataset of paired third and first person videos. arXiv preprint arXiv:1804.09626 (2018)
34. Soomro, K., Zamir, A.R., Shah, M.: Ucf101: A dataset of 101 human actions classes from videos in the wild. arXiv preprint arXiv:1212.0402 (2012)
35. Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., Metzler, D.: Long range arena: A benchmark for efficient transformers. arXiv preprint arXiv:2011.04006 (2020)
36. Kitaev, N., Kaiser, Ł., Levskaya, A.: Reformer: The efficient transformer. arXiv preprint arXiv:2001.04451 (2020)
37. Wang, S., Li, B., Khabsa, M., Fang, H., Ma, H.: Linformer: Self-attention with linear complexity. arXiv preprint arXiv:2006.04768 (2020)

38. Zaheer, M., Guruganesh, G., Dubey, A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al.: Big bird: Transformers for longer sequences. arXiv preprint arXiv:2007.14062 (2020)
39. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2018) 4510–4520
40. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is worth 16x16 words: Transformers for image recognition at scale. In: International Conference on Learning Representations (ICLR). (2020)
41. Carreira, J., Noland, E., Banki-Horvath, A., Hillier, C., Zisserman, A.: A short note about kinetics-600. arXiv preprint arXiv:1808.01340 (2018)
42. Tang, Y., Ding, D., Rao, Y., Zheng, Y., Zhang, D., Zhao, L., Lu, J., Zhou, J.: Coin: a large-scale dataset for comprehensive instructional video analysis. In: Conference on Computer Vision and Pattern Recognition (CVPR). (2019)
43. Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)