# A Overview

We report additional results in Section B including on the reconstruction task (Section B.1) and when using augmentations for downstream tasks (Section B.5). We visualise additional learned augmentations (e.g. brightness, saturation and rotation) in Section B.5 and give additional visualisations of the cropping augmentation. We also provide a video comparing our compressed representation to that of MPEG and visualising our learned augmentations.

We provide a more comprehensive study on the speed of various components of our setup in Section B.6 and additional information about the datasets in Section C. Section D has a more detailed exposition on WalkingTours – our new dataset and task for handling very long video sequences. Finally, we put together a more detailed description of our models in Section E, including how we apply S3D on our neural codes (Section E.1) as well as information about the other architectures, the precise hyperparameter sweeps we consider (Section E.3) and training details (Section E.4).

# B Additional Results

## B.1 Reconstructions at Various Compression Rates

We expand the results from the main paper with reconstruction performance at various compression rates in Table 5. As before, we can see that at the same compression rate, the reconstruction performance using JPEG encodings degrades much more quickly than when using our neural codes; for instance, we can compress inputs four times more using our neural compression than the JPEG compression.

## B.2 Compression Trade-offs

In addition to our main results, we have also experimented with various hyperparameters, showing different trade-offs.

**Space- vs Time-compression.** Alternatively to downsampling spatial resolution (space-compression) we could also downsample temporal resolution (time-compression). For that purpose, we use temporal striding in convolutional kernels. To keep the same compression rate, CR 236, we use the following setup. In space-compression, we compress 32-frames long video with the spatial resolution $256 - by - 256$ into a tensor of shape: $32 \times 16 \times 16 \times 2$ (time x width x height x number of vocabularies). In time-compression, we compress the same video into a tensor of shape: $4 \times 32 \times 32 \times 4$ (time x width x height x number of vocabularies). Space-compression yields the SSIM score 89.3, whereas time-compression 89.8. Both results are comparable though there are small qualitative differences in the decoded videos for individual videos.

**Codebook size vs number of codebooks.** VQ-VAE [42] uses a fixed number of codes in a single codebook. Here, for a fixed compression ratio, we can either

Table 5: **Reconstruction error for neural codes.** Additional results. We compare our approach to using JPEG encodings of the frames. We report three standard reconstruction metrics (PSNR, SSIM and the mean absolute error (MAE)) for training on Kinetics600 at different compression rates (CRs). For MAE, lower is better whereas for PSNR and SSIM [62], 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~90 | 34.8 | 94.8 | 0.013 |
| CR~180 | 32.6 | 92.3 | 0.016 |
| CR~236 | 30.8 | 89.8 | 0.019 |
| CR~384 | 30.0 | 88.4 | 0.019 |
| CR~768 | 29.0 | 85.4 | 0.022 |

increase number of codes in a single codebook or use different codebooks. Note that, if we decide to increase the number of codebooks twice, we are free to increase the number of codes quadratically as

$$c_r = \frac{I_T I_H I_W * 3 * \log_2 256}{T_T T_H T_W (2T_C) \log_2 K} = \frac{I_T I_H I_W * 3 * \log_2 256}{T_T T_H T_W T_C \log_2 K^2}$$

where $c_r$ is the compression rate, $I_H, I_W$ are spatial resolutions for each frame, $I_T$ is the number of frames in a single video, $T_T T_H, T_W$ denotes the shape of the compressed spatio-temporal tensor, $T_C$ is the number of codebooks and $K$ is the number of codes in each codebook (we need to store indices to these codes). Even though compression rates are the same, we found that it is better to increase the number of codebooks at the cost of fewer codes per codebook. That is, a single codebook with 65k codes gives worse results than two codebooks, with 256 codes each. The corresponding SSIM scores are 86 and 88.

### B.3 Using Compressed Representations versus Reconstructions

In the paper, we train downstream tasks directly using the neural codes. However, at the cost of significant speed loses as shown in Section B.6, we could take the neural codes, pass them through the generator $c^{-1}$ and obtain the reconstructed images. This generation / reconstruction process is akin to the decompression process in standard (non-neural) codecs. We could then use these reconstructed

images to train the downstream tasks. In Table 6, we compare the performance between using the neural codes and reconstructed images on Kinetics600. We find that there is a small drop in performance between using the neural codes and reconstructed images. This demonstrates that if the representation better captures the full image, we would expect improvements in performance (i.e. the drop in performance is not from the compression itself but from the quality of the learned representation).

Table 6: **Downstream classification accuracy on Kinetics600.** Additional results. We compare using the neural codes directly versus using the reconstructed images. We report Top-1 accuracy on K600 when using neural codes trained on either K600 or WalkingTours. We experiment with different levels of compression (different compression rates (CRs)). CR~1 denotes the upper bound of using the original RGB frames. Using the neural codes as opposed to the reconstructed images leads to a minor drop in performance ( 1%), demonstrating that improving the quality of the representation would directly improve performance.

| K600 | | WalkingTours | |
|---|---|---|---|
| CR | Top-1 ↑ | CR | Top-1 ↑ |
| **Original images** | | | |
| CR~1 | 73.1 | CR~1 | 73.1 |
| **Reconstructed images** | | | |
| CR~30 | 71.2 | CR~30 | 72.8 |
| CR~475 | 69.0 | CR~256 | 71.4 |
| **Neural codes** | | | |
| CR~30 | 72.2 | CR~30 | 71.3 |
| CR~475 | 68.2 | CR~256 | 68.4 |

### B.4 Transferability of Augmentations

Here, we consider whether the learned augmentations are transferable. That is, can we train the *neural compressor* and *augmentation network* on one dataset and evaluate it on another? We compare the flipping and cropping transformations when the *neural compressor* and *augmentation network* are trained on WalkingTours or Kinetics600. Note that in the main paper, the *neural compressor* was trained on WalkingTours and the *augmentation network* on Kinetics600. As can be seen, using either the flipping or cropping augmentation, we improve over the baseline setup that does not use learnt augmentations. This is valid even if the *neural compressor* and *augmentation network* are trained on different datasets than the classification model (which is trained on Kinetics600). Pre-training both the *neural compressor* and *augmentation network* on the same dataset as

the classification network sometimes improves performance, but the difference between setups is marginal. We did experiment with augmentations at larger compression rates but found that the results did not improve over the baselines; future work should explore how to learn augmentations at larger compression rates.

Table 7: **Using our learnt network for augmentation.** We report Top-1 accuracy on K600. We compare training the *neural compressor* (**C**) and *augmentation network* (**A**) on the same or different datasets than what the classification network is trained on. For each group, we bold the best setup that improves over the baseline setup (which uses no learned augmentations). As can be seen, we can train the *augmentation network* on WalkingTours, Kinetics600, or a combination thereof and still improve on the original *neural compressor*.

| | | 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) [44] | 224 | 60.7 | 63.0 | 68.1 | 69.0 |
| 256 central crop | 256 | 60.8 | 62.4 | 68.2 | 68.9 |
| **C: WalkingTours, A: Kinetics600** | | | | | |
| 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 |
| 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** |
| **C: WalkingTours, A: WalkingTours** | | | | | |
| Ours (2 spatial crops) | 224 | 61.2 | 64.1 | 69.2 | 69.2 |
| Ours (3 spatial crops) | 224 | **61.5** | 64.7 | **69.5** | 69.5 |
| Ours (4 spatial crops) | 224 | **61.5** | **64.8** | 68.9 | 69.5 |
| Ours (with flipping at train) | 256 | 61.8 | 63.9 | 68.3 | 69.5 |
| Ours (with flipping at train and eval) | 256 | **62.6** | **65.1** | **69.3** | **70.4** |
| **C: Kinetics600, A: Kinetics600** | | | | | |
| Ours (2 spatial crops) | 224 | 61.3 | 62.9 | **68.3** | 69.3 |
| Ours (3 spatial crops) | 224 | 61.3 | 63.2 | **68.3** | 69.2 |
| Ours (4 spatial crops) | 224 | **61.6** | **63.6** | 68.2 | 69.3 |
| Ours (with flipping at train) | 256 | 62.9 | 64.1 | **70.1** | **70.8** |
| Ours (with flipping at train and eval) | 256 | **63.5** | **64.7** | 70.0 | 70.4 |

### B.5 Extra Augmentation Visualisations

*Cropping augmentation.* We give additional visualisations of the cropping transformation in Figure 7. For a single image, we visualise four crops when that crop is applied (a) to the original image, (b) to the reconstructed image using our encoder-decoder network and (c) using our *augmentation network* applied to the neural codes. As can be seen, there is minimal difference between the crops using our *augmentation network* and the original image.

*Brightness augmentation.* In this work, we mainly focus on learning transformations of the latent space for the purpose of augmentations, and thus, we focus on the most common ones. However, we can also learn other transformations such as changes in brightness, as shown in Figure 6 for three videos at two brightness extremes. We have not found this transformation to improve performance of the classification models, so we show it here mainly to illustrate how our method is general. Note that, unlike when *cropping*, we cannot apply such a transformation by just manipulating neural codes (as in [44]).

Figure 8 compares the results for multiple videos. For a single image, four different amounts of brightness are applied. As in Figure 7, we visualise results when the transformation is applied to (a) the original image, (b) the reconstructed image using our encoder-decoder network and (c) using our *augmentation network* applied to the neural codes. Again, there is minimal difference between the results using our *augmentation network* and the original image.

*Other challenging transformations.* Figure 9 shows other challenging transformations: rotations and changes in saturation. We see that the *augmentation network* can successfully learn such transformations.

*Naive flips.* Finally, many transformations operating in the compressed space cannot be easily constructed. For instance, to flip frames in the video, we cannot flip the codes as every individual code, which corresponds to a spatial region, needs to also be flipped. Naively performing this operation leads to problematic artifacts after decoding as shown in Figure 10.

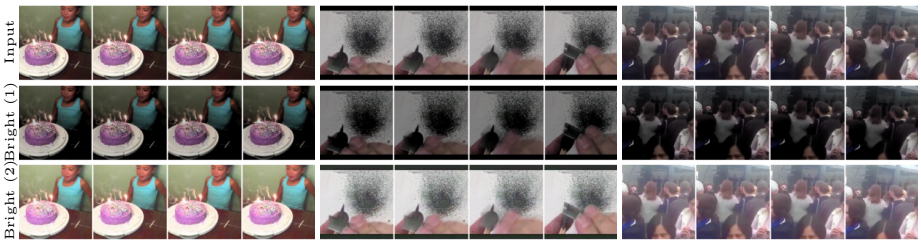


Fig. 6: **Learned augmentation: Brightness.** The top row shows the original frames for three videos; the bottom two rows show these frames after applying our equivariant network for brightness at two extremes.

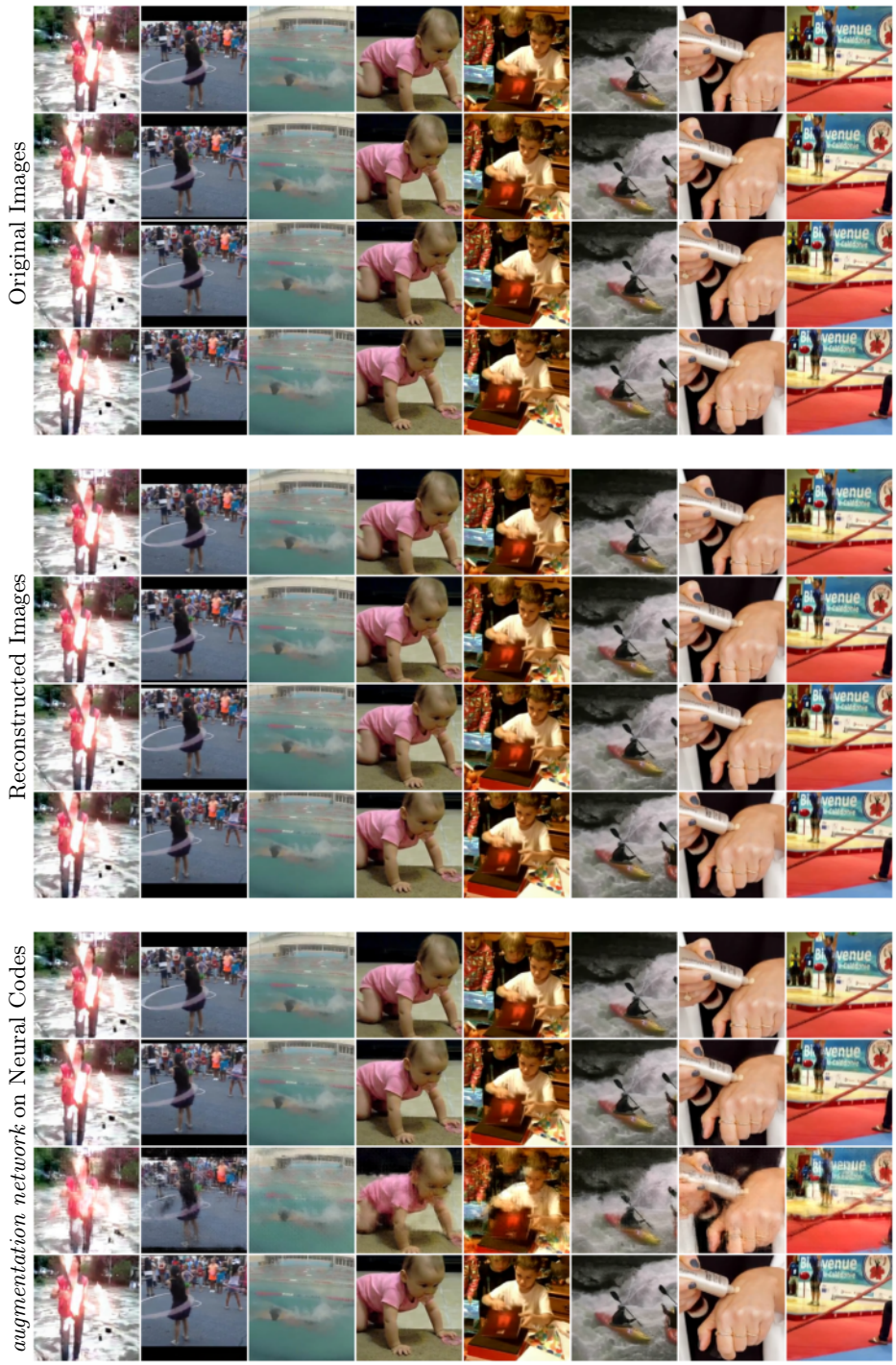


Fig. 7: **Learned augmentation: Cropping.**

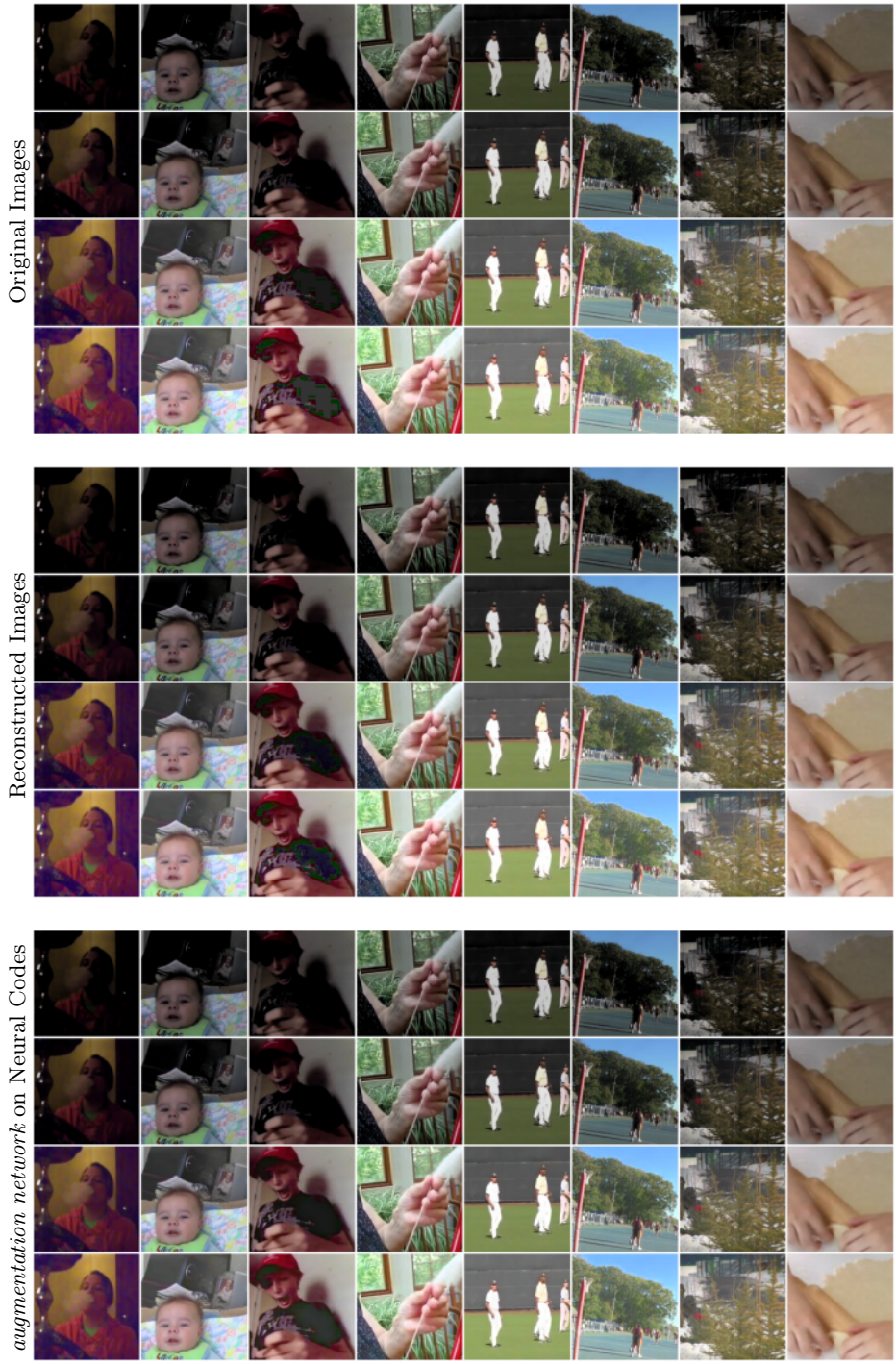


Fig. 8: **Learned augmentation: Brightness.**

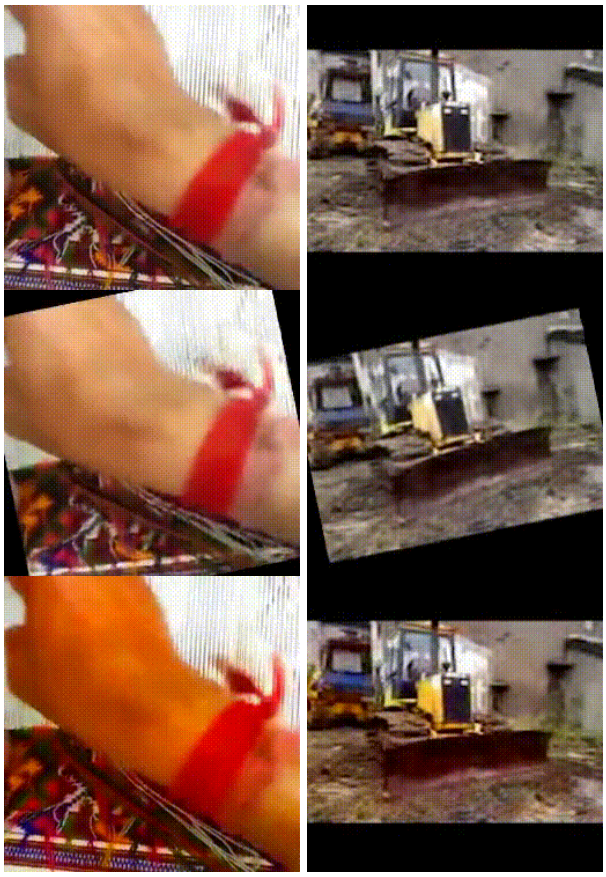

Fig. 9: **Learned augmentation: Rotations and Saturation.** Here, we show other, more challenging transformations. The top row presents the original video frames, middle row shows rotations whereas the bottom row saturation.

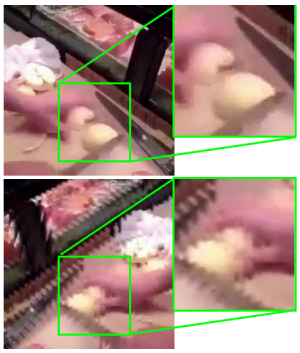


Fig. 10: Naively flipping neural codes leads to strong artifacts after decoding.

### B.6 Time Measurements

The compressed vision pipeline not only benefits from memory reduction, but also offers some speed gains. For instance, at compression rates of at least 256, we observe about 58% reduction in the inference time compared to using the RGB-valued inputs (assuming we only measure the speed cost of the classification network). We do not observe speed gains at lower compression rates, such as 30.

Table 8 gives detailed time measurements for the video inputs with 32 frames. More specifically, we measure the encoding and decoding times of our *neural compressor* as well as the augmentation time with our *augmentation network*. We also show the time measurements of the *augmentation network* together with the inference of the Kinetics model (augment & infer in the table). This is the most typical setting in our pipeline. This is because encoding can be done once in practice to store intermediary representation. Similarly, decoding is also not required in our pipeline, as we operate directly on the neural codes. This contrasts with standard vision pipelines that require decoding, and as we can see in Table 8, neural decoders incur a significant cost to the overall speed. Even JPEG decompression of a video with 32 frames, for the 30x compression rate and batch size 1 takes about 0.022s, which is slower than working directly with neural codes. Thus, it is more efficient to avoid this step, and to perform the required computation in the compressed space. Note that decoders are often implemented using larger networks than encoders. In contrast, our *augmentation network* is a relatively shallow network, described in Section E.1; this makes the whole pipeline efficient.

### B.7 Comparison to Other Methods

Here, we further discuss how our pipeline compares with other methods. We summarize these findings in Table 9. Other methods [1, 9, 35, 48, 65] operate directly on encoded frames (e.g. I-Frames, P-Frames, and/or residuals) or may use additional motion vectors that are either learned or obtained from an MPEG representation. That often requires devising new architectures; [35] needs to add extra lateral connections that fuse two pathways at different frame-rates, and [48] trains a flow together with the downstream task.

While [9, 65] use standard architectures (e.g. YOLO [46] or ResNets [27]), they require a large amount of engineering complexity in manipulating the MPEG codes into an appropriate input modality ([9] partially decode the MPEG codes in order to perform pixel-level predictions), whereas we directly use our compressed codes with *no* further data engineering. Moreover, [65] applies different models to different compressed representations yielding a codec-specific architecture.

Our compressed vision setup uses standard video architectures *without* requiring the development of an entirely new pipeline and we operate *directly* on the compressed codes with no further data engineering, and in a task-agnostic way. We also show how to perform augmentations directly in the compressed space. Moreover, in Section 4.2, we find that our compression method performs better than MPEG at different compression rates; thus showing the need for

Table 8: **Speed measurements.** We report the inference time of the classification network, the encoding / decoding time of our *neural compressor*, the augmentation time of our *augmentation network*, and the overall time of the combined augmentation and inference (augment & infer). We report, in seconds, time averaged over 100 runs. The standard deviation is marginal in all cases. We use Tesla V100. We use batch size 8 for RGB and compression rate 256 to better utilize vectorized computations, and batch size 1 for compression rate 30 as otherwise we run into memory issues (column BS for batch size). CR denotes different compression rates.

| **Measurement** | **BS** | **CR** | **Time** |
|---|---|---|---|
| Inference (rgb) | 8 | 1 | 0.089 |
| Inference (codes) | 8 | 256 | 0.052 |
| Encoding | 8 | 256 | 1.894 |
| Decoding | 8 | 256 | 3.076 |
| Augmentation | 8 | 256 | 0.078 |
| Augment & infer | 8 | 256 | 0.129 |
| Encoding | 1 | 30 | 0.816 |
| Decoding | 1 | 30 | 1.328 |
| Augmentation | 1 | 30 | 0.080 |
| Augment & infer | 1 | 30 | 0.102 |

trainable codecs. As our work is *not* MPEG specific, it could potentially benefit from better compressed representations.

## C Datasets

Here, we provide more information about the datasets.

*Kinetics600* consists of short video clips downloaded from YouTube. The task is to predict which action corresponds to a given video, which is formalized as a classification problem (out of 600 classes) given the whole video. However, these clips are up to 10 second long, and most video models are trained on about 2 second long clips at 25fps. Kinetics 600 has about 400k video clips for training purpose, and 100k video clips for evaluation, with less than 60 days of total video time. Nonetheless, it is a popular benchmark in the research community, and even though our compressed vision framework is not essential, it can still make the whole training time much more efficient.

*COIN* consists of relatively long video clips of the order of a couple of minutes; on average 2 minutes and 36 seconds. The dataset has 476 hours of total video time. These are instructional videos with annotations per frame describing the task being shown in the frame. It has about 12k video clips. We evaluate our

Table 9: Comparison of our pipeline to other methods. We compare whether each method uses an MPEG style codec (e.g. I-Frames or Blocks from that representation), a flow (optical flow, motion vectors, or their approximations), or whether the method leverages standard video pipelines (existing popular architectures and augmentations). Some methods operate on MPEG style representations. Thus they require new architectures, training schemes, or data engineering. In contrast, our pipeline can directly be used with the existing video architectures, and we can train the corresponding augmentations. Finally, other approaches rely on an MPEG style representation that lead to worse representation at higher compression rates to ours as demonstrated in Section 4.2.

| Method | no MPEG | no flow | standard pipelines |
|---|---|---|---|
| Alizadeh et al. [1] | ✗ | ✗ | ✗ |
| DMC-Net [48] | ✗ | ✗ | ✗ |
| Li et al. [35] | ✗ | ✗ | ✗ |
| Wu et al. [65] | ✗ | ✓ | ✗ |
| Chen et al. [9] | ✗ | ✓ | ✓ |
| Ours | ✓ | ✓ | ✓ |

approach on its ability to perform per-frame annotations as opposed to per-video classification as in Kinetics600.

*WalkingTours.* As we could not find any dataset containing sufficiently long videos, we decided to collect our own dataset as a proof of concept showing the utility of our proposed pipeline. The dataset consists of one-hour long videos of tourists walking in different places, and it poses various challenges for sampling and processing very long video sequences. These include high memory requirements for storage, high bandwidth, distributed and asynchronous requirements for sampling data and sending them to device, and high memory consumption on a device like GPU due to processing very large volumes of data points. Before even defining the right task on long video understanding, first these challenges above should be addressed.

The dataset has about 18k videos, with 1815 videos used for validation and the same number for a held-out test; the rest is the training set. The videos range in length from 18 minutes to ten hours; on average 40 minutes. In total, the dataset amounts to around 500 days worth of accumulated video time. It is significantly more than the number of days of accumulated video time in the latest Kinetics dataset [31] and other egocentric datasets [10, 11, 22]. However, WalkingTours has no human annotations, and currently it is only compatible with self-supervised or unsupervised training or evaluation.

## D Walking Tours: Task and Components

*Description.* WalkingTours is a dataset of very long, even one-hour long, videos. In the current form, it is also a purely visual dataset, i.e. there are no extra

annotations associated with it; hence, models can only be trained and evaluated in an unsupervised way. For this purpose, we have proposed a continual setting, where hour-long videos are split into many 5s-long video clips that the model observes in sequence, one by one. All these short clips belong the the same video, which is continuous (has no cuts), and we call them chunks. At each step, whenever the network sees a new chunk, we randomly sample a chunk from anywhere in the whole video and pass it to the network as a query. This clip might belong to the part of the video that the network has already seen, i.e. its *past*, or a part that has not been yet observed, i.e. its *future*. We name our task *Past-Future*, and illustrate the whole setup in Figure 11. As can be seen, to solve this task, the network can use memory of the past events. To avoid pure memorization, we always randomly and spatially crop the clips, so that the query is almost surely different than the past chunks. For training, we only do backpropagation over these chunks individually, and aggregate all such gradients together for the update step. In the following, we describe all the components, *memory, adapter, core* and *predictor*, that we use in our experiments on this task.

*Memory.* Our task requires efficient memory usage. Here, we investigate that angle by comparing *LSTM*, *Slot* and using no memory (*none*). A network without memory needs to respond to visual stimuli reactively, based only on the current observations. In our task, such network should perform at random chance (50-50), which is confirmed experimentally (52.9%). *LSTM* is the most popular recurrent neural network, equipped with gating operations that enable long short-term memory usage [28]. Although LSTMs work well with shorter video sequences [13, 32, 36, 51, 53], working with longer videos become more problematic due to interference [2, 20, 21, 37]. This is also the case here. The network is only able to learn the following simple strategy. On the question if the query is in the *past*, it answers negatively for the first half of the video, and positively for the second half of the video. Note that, after observing almost the whole video the answer is very likely positive. This simple strategy yields 78.2%. *Slot* is an external memory unit [23, 41, 63, 64] that can explicitly store past representations. In our study, we employ a deterministic writing operation, where a new memory is added to old memories whenever chunk $t$ is observed, i.e., $\mathcal{M}^{t+1} := \{\boldsymbol{m}^t\} \cup \mathcal{M}^t$. $\mathcal{M}^t$ represents all the slots at time step $t$. The reading operation is learned using a specialized neural network. The network equipped with the *slot* memory can solve the task on the half-an-hour long video understanding (99.5%).

*Adapter.* Adapter is a shallow 3D CNN that operates on a chunk creating the task-specific representation. It also reduces the dimensionality of the input. The same as other our experiments, chunks are already neural codes. Without compressing the inputs into such representations, it became challenging to even sample data points and transfer them through a bandwidth-limited network.

*Core.* It is a neural network that interacts between queries and memories. Here, we experiment with a cross-source transformer, which uses the query as a transformer-query $q$ and memory elements as transformer-key $k$ and -values $v$ – adopting

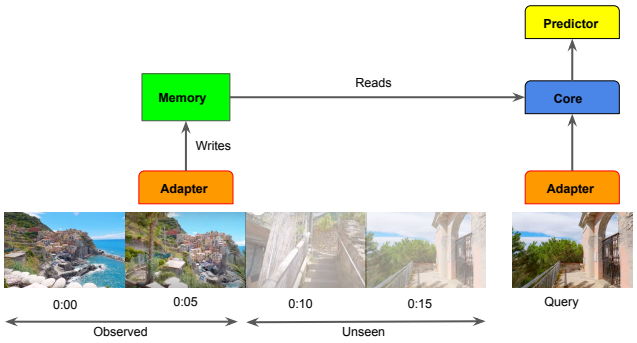


Fig. 11: Our online learning framework. We use the same colours to indicate weights sharing. Here, the query comes from the *future*.

naming convention from the transformer literature [58] – or it can also be seen as an asymmetric variant of a cross-modal transformer [57]. Note that, although standard transformer operate on the whole input sequence, here, the memory is a bottleneck between the input signal and the transformer. Moreover, due to the asymmetry, the network's cross-attention scales better than pure attention to longer sequences.

*Predictor.* It is a single linear layer that outputs a scalar describing whether the query is in the *past* or *future*. Overall, to train networks on very long videos, we need to trade-off the network's complexity with its capacity. Thus, we use relatively simpler architecture like a shallow 3D CNNs in Adapter.

## E Modelling

This section discusses architectures and training protocols in more detail. In particular, we describe the recognition architecture, architectures used to implement *neural compressor* and *augmentation network*, and our hyper-parameters.

### E.1 Architecture

**S3D.** As compressed embeddings have smaller spatial dimensions than the RGB-valued inputs, we adapted the striding values of the S3D architecture so that the shapes of the internal tensors, between models operating on RGB and neural codes, are roughly the same. Figure 12 reproduces Figure 6 from [66]; it shows how the S3D architecture is applied to a standard video input. We then show the modifications to the strides and output channels that we use in S3D when operating on neural codes at CR$\sim$ 30 in the main paper; we visualise these changes in Figure 13. Figure 14 visualises the changes if we have a larger

compression rate and obtain codes that have a width and height of about one eighth the size of the original image. We use this setup for CR~ 256 and CR~ 475 in the main paper. In general, applying S3D to our neural codes requires only a few small changes: modifying the strides of the input convolution and some of the max pool layers as well as modifying the output channels of the first two convolutional layers.

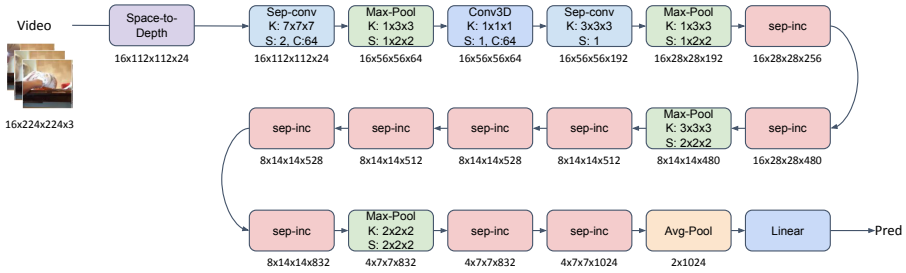


Fig. 12: **S3D.** Reproduction of Figure 6 from [66]. This shows how the standard S3D architecture is applied to a video. Note that we show the result after first applying a space to depth transformation to the input. Below each layer, we write the size of the output tensor for the given input size.

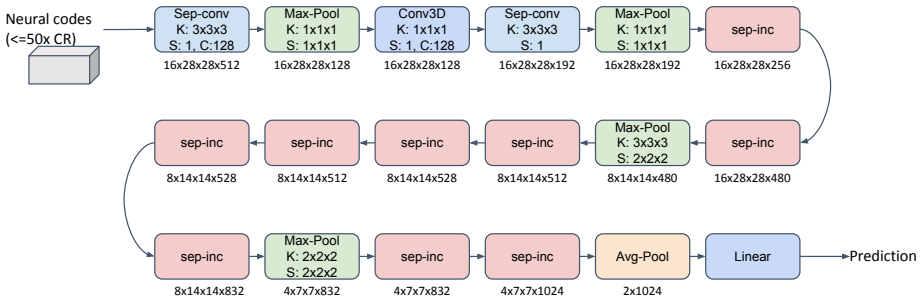


Fig. 13: **How we modify the standard S3D architecture for smaller compression rates.**. Below each layer, we write the size of the output tensor for the given input size. In comparison to Figure 12, we only change the strides of the first convolution, the first two max pools and modify the output channels in the first two convolutional layers.

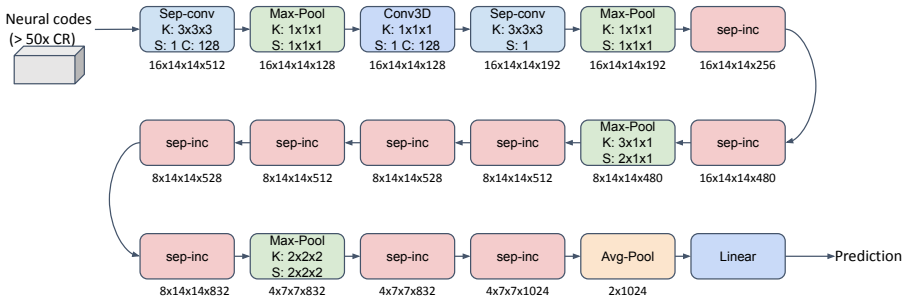


Fig. 14: **How we modify the standard S3D architecture for larger compression rates.**. Below each layer, we write the size of the output tensor for the given input size. In comparison to Figure 12, we only change the strides of the first convolution, the first three max pools and modify the output channels in the first two convolutional layers.

**Encoder-Decoder.** The actual setup of the two components of the architecture is given in Table 10. Note that in order to generalise to different numbers of frames, we have to reflect the tensor at the boundary when padding (as opposed to padding with zeros).

*Encoder $c$.* The encoder consists of $E_s$ ResNet blocks that make the resolution smaller. This is followed by $E_c$ ResNet blocks that maintain the spatial resolution size.

*Decoder $c^{-1}$.* The decoder consists of $D_c$ ResNet blocks that maintain the spatial resolution size. This is followed by $D_s$ ResNet blocks that increase the resolution to the original image resolution size.

*Quantized embeddings.* To create quantized embeddings, we take the $V_e$ channels and split them into $N$ codebooks. These are the embeddings we use to perform nearest neighbours and find the corresponding embedding in embedding space.

**Augmentation network.** Our *augmentation network* has two main components: MLP and Transformer.

*MLP.* The MLP takes the input augmentation conditioning values, e.g. describing bounding box coordinates or brightness values or whether flipping happened, which are flattened to form a single vector; these are passed to three hidden layers of size 64. The output is a vector with the same number of channels as the neural code. We broadcast the embedding from the MLP along the spatial dimensions and concatenate it with the neural code to give a tensor with double the size of the latter.

*Transformer.* The transformer takes as input the concatenated tensor. The transformer has two hidden layers of size 128, 4 heads and uses an absolute positional embedding.

### E.2 VQ-VAE and Compression Rates

With the VQ-VAE encoding-decoding scheme, we can indirectly control compression rates as follows. First, we use 3D CNNs with $L_c$ layers that downscale spatial dimensions of the input tensor $H_i \times W_i$ to form a tensor of shape $H_c \times W_c$. As we are using striding two, new dimension $H_c$ is $2^{L_c}$ times smaller than the corresponding dimension of the input frame. We do not compress along the time dimension. As we are using $N_c$ codebooks, the neural codes form a tensor with the shape $H_c \times W_c \times N_c$. Each element of such a tensor is a discrete number indicating which embedding in the corresponding codebook is used for the reconstruction. The larger codebook, the more bits per such an element are needed. Thus the final compression ratio is: $c_r = \frac{H_i \times W_i \times 3 \times \log_2(256)}{H_c \times H_w \times N_c \times \log_2(K_c)}$ where $K_c$ is the number of the codebook's elements.

Table 10: **Model Architecture.** The model architecture of the encoder-decoder models for an input video with $F$ frames of size $H, W, C$ ($C$ may be greater than 3 if we are using spatio temporal crops). $s$ denotes stride while $k$ denotes the kernel size. $oc$ denotes the number of output channels and $i$ the block index within the group.

| ID | Output Size | Block |
|---|---|---|
| Encoder | $F \times H_c \times W_c \times V_e$ | $\left\{\begin{array}{c}\underline{\text{Conv3D}} \\ s = (1,2,2) \\ k = (4,4,4) \\ oc = V_e/2^{E_s - i - 1} \\ \text{ReLU}\end{array}\right\} xE_s$ |
| Encoder | $F \times H_c \times W_c \times V_e$ | $\left\{\begin{array}{c}\underline{\text{Conv3D}} \\ s = (1,1,1) \\ k = (3,3,3) \\ oc = 4V_e \\ \text{ReLU} \\ \underline{\text{Conv3D}} \\ s = (1,1,1) \\ k = (3,3,3) \\ oc = V_e \\ \text{ReLU}\end{array}\right\} xE_c$ |
| Decoder | $F \times H_c \times W_c \times V_e$ | $\left\{\begin{array}{c}\underline{\text{Conv3D}} \\ s = (1,1,1) \\ k = (3,3,3) \\ oc = 4V_e \\ \text{ReLU} \\ \underline{\text{Conv3D}} \\ s = (1,1,1) \\ k = (3,3,3) \\ oc = V_e \\ \text{ReLU}\end{array}\right\} xD_c$ |
| Decoder | $F \times H \times W \times C$ | $\left\{\begin{array}{c}\underline{\text{Conv3DTranspose}} \\ s = (1,2,2) \\ k = (4,4,4) \\ oc = V_e/2^i \\ \text{ReLU}\end{array}\right\} x(D_s - 1)$ |
| Decoder | $F \times H \times W \times C$ | $\left\{\begin{array}{c}\underline{\text{Conv3DTranspose}} \\ s = (1,2,2) \\ k = (4,4,4) \\ oc = C\end{array}\right\}$ |

### E.3 Sweeps

In order to choose the best hyperparameters for the compression architecture we swept over the following model choices. For a given setup, we chose the set of hyperparameters with the best performance on the validation set.

#### Encoder-decoder architecture

*Input transformation.* First, we swept over the type of spatio temporal patches. We considered two setups: we can either use the original shape of the tensor or we can reshape the tensor by dividing it into temporal and spatial crops. We considered either using the original video shape or dividing the video into four spatial crops and concatenating along the channel dimension. However, we found that taking further crops in either the spatial or temporal dimension hurt performance substantially, so did not investigate further.

*Architecture size.* We swept over the number of encoder ResNet blocks $E_s$ (we considered [3, 4, 5]) and corresponding decoder ResNet blocks $D_s$ (again [3, 4, 5]) that change the final spatial size of the embedding. We also swept over the number of intermediary ResNet blocks $E_c, D_c$ in the encoder and decoder (we swept over using [3, 5, 7] blocks for both). Here, we found as in [38], that using more decoder blocks improves reconstruction performance (as opposed to more encoder blocks).

*Codebook.* We swept over the number of codebooks (we considered one or two), the number of embeddings in the codebooks (we consider [256, 512, 1024, 4096, 8192]) and the size of those embeddings (we consider embeddings of size [128, 256, 512]).

*Training parameters.* Finally, we swept over the learning rate when training the encoder-decoder model. We considered learning rates [3e-4, 1e-5] and used an SGD optimizer.

#### Augmentation Network

*MLP.* We swept over the number of hidden dimensions and the size of those dimensions: we considered [[128], [128, 128], [64, 64, 64]]. In general, we found using more hidden layers performed better.

*Transformer.* We swept over the number of heads ([1, 4]) and the size of the intermediate representation ([128, 256]). In general, these choices did not make a large difference in the results.

### E.4 Training.

Our pipeline consists of three training stages, which we detail below.

**Neural compressor**. The *neural compressor* was trained until convergence on either the Kinetics600 or WalkingTours dataset. To chose the best configuration of parameters, we swept over the hyperparameters (as described above) and selected the one with the best reconstruction loss on the validation set. The precise sweeps are given in Section E.3.

**Augmentation network**. The *augmentation network* was trained until convergence using a learning rate of 0.001, the Adam optimizer and no weight decay.

**Downstream training.** Finally, the downstream networks were trained using the Adam optimizer with cosine decay and an initial learning rate of 0.5 and weight decay of 1e−5. The models were trained for 60 epochs on COIN and 135 epochs on Kinetics600.

# Bibliography


[1] Alizadeh, M., Sharifkhani, M.: Compressed domain moving object detection based on crf. IEEE Transactions on Circuits and Systems for Video Technology (2019)

[2] Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., Tuytelaars, T.: Memory aware synapses: Learning what (not) to forget. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 139–154 (2018)

[3] 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. pp. 3286–3295 (2019)

[4] Bertasius, G., Wang, H., Torresani, L.: Is space-time attention all you need for video understanding? arXiv preprint arXiv:2102.05095 (2021)

[5] Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)

[6] Carreira, J., Noland, E., Banki-Horvath, A., Hillier, C., Zisserman, A.: A short note about kinetics-600. arXiv preprint arXiv:1808.01340 (2018)

[7] 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)

[8] 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)

[9] Chen, L., Sun, H., Katto, J., Zeng, X., Fan, Y.: Fast object detection in hevc intra compressed domain. In: 2021 29th European Signal Processing Conference (EUSIPCO) (2021)

[10] Damen, D., Doughty, H., Farinella, G.M., Fidler, S., Furnari, A., Kazakos, E., Moltisanti, D., Munro, J., Perrett, T., Price, W., Wray, M.: Scaling egocentric vision: The epic-kitchens dataset. In: European Conference on Computer Vision (ECCV) (2018)

[11] Damen, D., Doughty, H., Farinella, G.M., Furnari, A., Kazakos, E., Ma, J., Moltisanti, D., Munro, J., Perrett, T., Price, W., et al.: Rescaling egocentric vision. arXiv preprint arXiv:2006.13256 (2020)

[12] DeVries, T., Taylor, G.W.: Dataset augmentation in feature space. arXiv preprint arXiv:1702.05538 (2017)

[13] Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., Darrell, T.: Long-term recurrent convolutional networks for visual recognition and description. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2625–2634 (2015)

[14] 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)

[15] Dubois, Y., Bloem-Reddy, B., Ullrich, K., Maddison, C.J.: Lossy compression for lossless prediction. In: Advances in neural information processing systems (NeurIPS) (2021)
[16] Ehrlich, M., Davis, L.S.: Deep residual learning in the jpeg transform domain. In: Proceedings of International Conference on Computer Vision (ICCV) (2019)
[17] Esser, P., Rombach, R., Ommer, B.: Taming transformers for high-resolution image synthesis. arXiv preprint arXiv:2012.09841 (2020)
[18] Fan, H., Xiong, B., Mangalam, K., Li, Y., Yan, Z., Malik, J., Feichtenhofer, C.: Multiscale vision transformers. arXiv preprint arXiv:2104.11227 (2021)
[19] Feichtenhofer, C., Fan, H., Malik, J., He, K.: Slowfast networks for video recognition. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 6202–6211 (2019)
[20] French, R.M.: Catastrophic forgetting in connectionist networks. Trends in cognitive sciences **3**(4), 128–135 (1999)
[21] Goodfellow, I.J., Mirza, M., Xiao, D., Courville, A., Bengio, Y.: An empirical investigation of catastrophic forgetting in gradient-based neural networks. arXiv preprint arXiv:1312.6211 (2013)
[22] Grauman, K., Westbury, A., Byrne, E., Chavis, Z., Furnari, A., Girdhar, R., Hamburger, J., Jiang, H., Liu, M., Liu, X., et al.: Ego4d: Around the world in 3,000 hours of egocentric video. arXiv preprint arXiv:2110.07058 (2021)
[23] Graves, A., Wayne, G., Danihelka, I.: Neural turing machines. arXiv preprint arXiv:1410.5401 (2014)
[24] 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. pp. 6047–6056 (2018)
[25] 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)
[26] Härkönen, E., Hertzmann, A., Lehtinen, J., Paris, S.: Ganspace: Discovering interpretable gan controls. Advances in Neural Information Processing Systems (NeurIPS) (2020)
[27] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
[28] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
[29] 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. pp. 603–612 (2019)
[30] Jahanian, A., Chai, L., Isola, P.: On the "steerability" of generative adversarial networks. In: International Conference on Learning Representations (ICLR) (2020)

[31] 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)
[32] Kim, C., Li, F., Rehg, J.M.: Multi-object tracking with neural gating using bilinear lstm. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 200–215 (2018)
[33] Kitaev, N., Kaiser, Ł., Levskaya, A.: Reformer: The efficient transformer. arXiv preprint arXiv:2001.04451 (2020)
[34] 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)
[35] Li, J., Wei, P., Zhang, Y., Zheng, N.: A slow-i-fast-p architecture for compressed video action recognition. In: Proceedings of the 28th ACM International Conference on Multimedia (2020)
[36] Li, Z., Gavrilyuk, K., Gavves, E., Jain, M., Snoek, C.G.: Videolstm convolves, attends and flows for action recognition. Computer Vision and Image Understanding **166**, 41–50 (2018)
[37] McCloskey, M., Cohen, N.J.: Catastrophic interference in connectionist networks: The sequential learning problem. In: Psychology of learning and motivation, vol. 24, pp. 109–165. Elsevier (1989)
[38] Mentzer, F., Toderici, G.D., Tschannen, M., Agustsson, E.: High-fidelity generative image compression. In: Advances in Neural Information Processing Systems (NeurIPS) (2020)
[39] Mnih, A., Gregor, K.: Neural variational inference and learning in belief networks. In: International Conference on Machine Learning. pp. 1791–1799. PMLR (2014)
[40] 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)
[41] Oh, S.W., Lee, J.Y., Xu, N., Kim, S.J.: Video object segmentation using space-time memory networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9226–9235 (2019)
[42] Oord, A.v.d., Vinyals, O., Kavukcuoglu, K.: Neural discrete representation learning. arXiv preprint arXiv:1711.00937 (2017)
[43] 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)
[44] 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)
[45] 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)
[46] Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR). pp. 779–788 (2016)

[47] 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. pp. 4510–4520 (2018)
[48] Shou, Z., Lin, X., Kalantidis, Y., Sevilla-Lara, L., Rohrbach, M., Chang, S.F., Yan, Z.: Dmc-net: Generating discriminative motion cues for fast compressed video action recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR) (2019)
[49] 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)
[50] 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)
[51] Srivastava, N., Mansimov, E., Salakhudinov, R.: Unsupervised learning of video representations using lstms. In: International conference on machine learning. pp. 843–852. PMLR (2015)
[52] 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. pp. 625–634 (2020)
[53] Sun, L., Jia, K., Chen, K., Yeung, D.Y., Shi, B.E., Savarese, S.: Lattice long short-term memory for human action recognition. In: Proceedings of the IEEE international conference on computer vision. pp. 2147–2156 (2017)
[54] 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)
[55] 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)
[56] 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. pp. 4489–4497 (2015)
[57] Tsai, Y.H.H., Bai, S., Liang, P.P., Kolter, J.Z., Morency, L.P., Salakhutdinov, R.: Multimodal transformer for unaligned multimodal language sequences. In: Proceedings of the conference. Association for Computational Linguistics. Meeting. vol. 2019, p. 6558. NIH Public Access (2019)
[58] 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)
[59] Walker, J., Razavi, A., Oord, A.v.d.: Predicting video with vqvae. arXiv preprint arXiv:2103.01950 (2021)
[60] Wang, S., Li, B., Khabsa, M., Fang, H., Ma, H.: Linformer: Self-attention with linear complexity. arXiv preprint arXiv:2006.04768 (2020)
[61] Wang, X., Girshick, R., Gupta, A., He, K.: Non-local neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 7794–7803 (2018)
[62] Wang, Z., Bovik, A., Sheikh, H., Simoncelli, E.: Image quality assessment: from error visibility to structural similarity. IEEE Transactions on Image Processing **13**(4), 600–612 (2004). https://doi.org/10.1109/TIP.2003.819861

[63] Weston, J., Chopra, S., Bordes, A.: Memory networks. arXiv preprint arXiv:1410.3916 (2014)
[64] Wu, C.Y., Feichtenhofer, C., Fan, H., He, K., Krahenbuhl, P., Girshick, R.: Long-term feature banks for detailed video understanding. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 284–293 (2019)
[65] 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). pp. 6026–6035 (2018)
[66] Xie, S., Sun, C., Huang, J., Tu, Z., Murphy, K.: Rethinking spatiotemporal feature learning for video understanding. arXiv preprint arXiv:1712.04851 **1**(2), 5 (2017)
[67] 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)
[68] Yan, W., Zhang, Y., Abbeel, P., Srinivas, A.: Videogpt: Video generation using vq-vae and transformers. arXiv preprint arXiv:2104.10157 (2021)
[69] 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)