Image Encoding Schemes for Classification

Abhipsha Das* ad6489@nyu.edu

Alexander Miller* ahm9968@nyu.edu

Rhitvik Sinha* rs8438@nyu.edu

Abstract

First popularized for Natural Language Processing tasks, the Transformer architecture has spread to other domains including Computer Vision. The default setup for the Vision Transformer architecture splits the input into patches and then applies a linear transformation to each patch. Since then, researchers have tried other approaches such as applying convolutions before the transformer layers, vector quantization, mixed-resolution tokenization, and more. We survey techniques and compare their effect on parameter counts, runtime, and final performance. We conclude that the baseline itself – patch embeddings – offers an impressive tradeoff when it comes to computational cost vs performance and is surprisingly effective despite its limited capacity. The code for our experiments is available at `https://github.com/alexholdenmiller/bagel`.

1 Introduction

The transformer architecture was originally proposed for language modeling applications [Vaswani et al., 2023]. Language modeling starts from discrete inputs (typically text represented as sequences of characters) which are converted to continuous representations using tokenization followed by a lookup table which stores an embedding vector for each unique token. Tokenization strategies vary from full words, partial words using subword strategies like BytePair Encoding [Sennrich et al., 2016] or WordPiece [Devlin et al., 2019], single characters [Zhang et al., 2016], or even bytes [Wang et al., 2019, Yu et al., 2023]. Typically, a positional encoding scheme is then used to modify the embeddings (for example, by adding a learned embedding based on the absolute position). Finally, the sequence of embeddings is fed into the transformer layers to complete semantic reasoning and accomplish the task at hand.

The exact choice of this approach can have a significant impact on the downstream performance. Word-based approaches suffer from "out-of-vocabulary" issues because rare words may appear in the test set but not in the training set, and typically reserve and "out of vocabulary" token to store meaning for rare words. Character-based or byte-based approaches do not have this issue, but the modeling problem becomes much more difficult because a model needs to be able to compose the full meaning of words from the meanings of the constituent characters rather than being able to store the meaning of the full concept contained in the word on its own. Subword approaches attempt to bridge this gap by looking for either the most frequent (BytePair) or most informative (WordPiece) subword units. This results in learned units of meaning that are more informative than single characters and require less composition by the model. This includes many full words: for example, LLaMA's [Touvron et al., 2023] tokenizer has the words "has", "the", and "words" as full tokens, but "token" and "izer" are two separate tokens – a useful construction since the former can be used with many different endings and the latter can be added to many different words. However, poor splits can also be a detriment to the algorithm's performance: see Figure 1 for an example of bad tokenization for mathematical statements.

*Equal Contribution; Department of Computer Science, Courant Institute of Mathematical Sciences, New York University

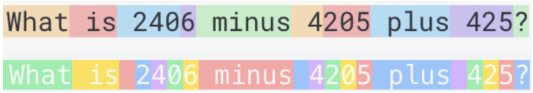


Figure 1: GPT3's [Brown et al., 2020] tokenizer (top) recognizes subwords within the numbers themselves, meaning the model has to memorize unique meaning for these numbers and learn to compose them depending on the ordering in which they appear. LLaMA's tokenizer (bottom), by contrast, does not form multi-digit subwords in order to always compose numbers from the individual digits themselves.

Other domains have unique perceptual challenges to overcome. Rather than an input of discrete units, other tasks feature continuous input data such as sequences of frequencies (audio) or two-dimensional grids of brightness (images) with multiple channels (mono vs stereo or grayscale vs color). While there is a similar need to reshape the input into the "expected" shape for a transformer, that is, a sequence of embeddings, the practitioner has considerable freedom in how to conduct this reshaping, including potentially devoting significant portions of the network parameter- or compute-capacity to this initial processing. This choice has significant ramifications for how the network understands the data (different priors on the transformation), the final performance of the network, and the data- and compute-efficiency of the approach.

2 Method

We explore several different strategies for initial processing of input image data before feeding it into a transformer trunk that we keep fixed-size across all different approaches.

In each approach, we prepend a "class token" to the sequence produced by the "tokenization" algorithm and then add learned absolute positional encoding embeddings to each value in the sequence.

2.1 Patch Embedding

We first try the default configuration for Vision Transformers: Patch Embeddings. In this approach, the image is split into fixed-size squares and then each patch is flattened and linearly transformed before the sequence of patches are fed into the network. This is easily implemented as a 2D convolution where the stride is equal to the kernel size.

The size of the patches (in particular, relative to the image size) can be adjusted in order to tune the "level of detail" of the model. Splitting the image into many, small patches allows the model to learn to recognize small local patterns but is more costly computationally. Fewer, larger patches can be processed more quickly, but may be too large to easily learn interesting patterns from the data.

2.2 Initial Convolutions

A natural alternative to the Patch Embedding is simply to apply normal convolutions. We explore a variety of different numbers of convolutional layers, kernel sizes, and stride lengths. Greater strides increases computational speed and low values here may not improve performance despite being costly. Smaller kernel sizes may make it easier to learn very local patterns, while larger ones slightly reduce the final output dimensions and thus result in a shorter sequence length for the transformer to process and slightly save computation time on that side while costing more in computation during the convolution itself. More convolutional layers drastically reduce the length of the sequence that is fed into the transformer and thus the total computation time of the model, as each convolution downsamples the input, while adding significantly to the total parameter count of the model.

2.3 Vector Quantization

After either applying the patch embedding or initial convolutions followed by a flattening, the model will now have a sequence of embeddings that could be fed into the transformer. However, a step that could be applied here before the transformer is vector quantization. Conceptually, one maintains a "codebook" of concepts (referred to as "codes") that may represent different concepts your model may

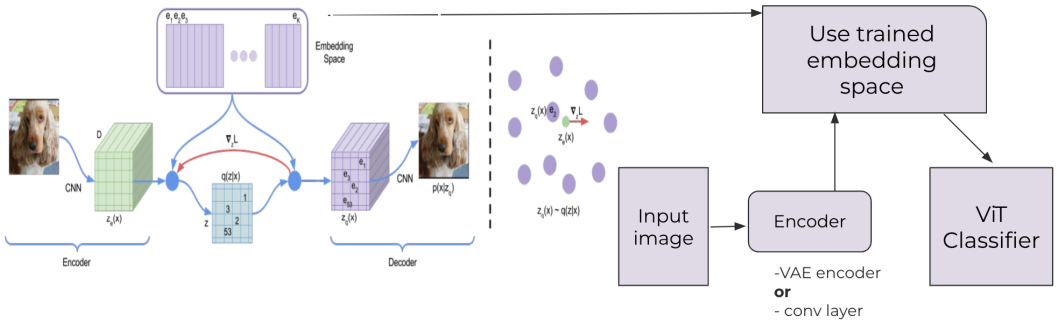


Figure 2: VQ-VAE Encoding for ViT

recognize in the input. Each input embedding is mapped to a single code in the codebook, i.e. the one which most closely resembles that input embedding. In fact, you can maintain a many codebooks, each containing different sets of codes, and inputs are mapped to one code from each codebook which are then combined back into a sequence of representations for the transformer that is the same size as it was before the quantization step.

Codes are selected using a gumbel softmax with an annealed temperature during training or simply an argmax during inference.

2.4 VQ-VAE

A VQ-VAE van den Oord et al. [2018] architecture is trained on reconstruction tasks and the trained encoder and vector quantize layer are used to obtain discrete latent representations or codes through a novel parameterisation of the posterior distribution of (discrete) latents given an observation, which is then classified using a vision transformer. The vector quantize layer calculates discrete latent variables using the shared embedding space and assigns discrete codes to represent the latent in terms of its nearest neighbours. The foward computations maps the latents to a one hot encoded vector of the size of the discrete space, The combined set of parameters of this model are the encoder, decoder, and the embedding space. For our experiments with a ViT, we are concerned with the encoder and embedding space after training the VQ-VAE.

VQ-VAE learns a single codebook for all categories of images, and utilizes a single partition for the latent space to assign each image feature an exclusive discrete representation. Figure 2 shows the training of the VQ-VAE and the how the trained codebook space is leveraged for capturing image features for classification tasks.

2.5 Mixed-Resolution Tokenization

Figure 3: Mixed Resolution Tokenization

As discussed above, a default Vision Tranformer models process input images by dividing them into a spatially regular grid of equal-size patches. In contrast, we utilize Mixed Resolution Tokens Ronen et al. [2023], splitting the image into a patch mosaic (see Figure 3) according to a saliency scorer, and employ a standard Transformer architecture with 2D position embeddings. This 'saliency score' for a patch p is calculated as:

$$Score_{feat}[p] = MSE(feat(im_{blur})[p], feat(im)[p])$$

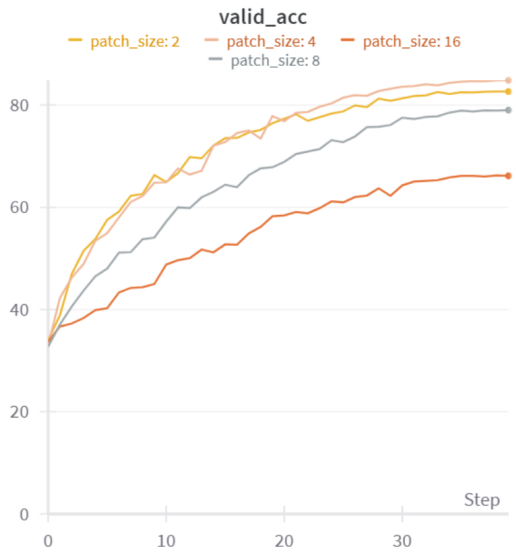


Figure 4: Patch size 4 outperforms 2, 8, and 16 and isn't too expensive to run.

where, $feat$ is a CNN (`torchvision` distribution of a `ShuffleNet-V2` Zhang et al. [2017], Ma et al. [2018] with ImageNet pre-trained weights), im is an RGB image, and im_{blur} is the blurred version of im.

Hence, $Score_{feat}$ estimates the semantic information loss from decreasing the resolution of an image patch by comparing its original representation to its representation in a blurred image. Intuitively, this score estimates how much semantic information is lost when we downsample an image patch.

3 Results

We evaluate most of our models by training for ∼40 epochs on CIFAR10 Krizhevsky and Hinton [2009] and then comparing validation accuracy. We fix image size to 32x32 for most images for ease of quickly training. We use the OneCycle learning rate scheduler with Adam / AdamW, a learning rate of 0.0006, weight decay of 0.1, batch size of 128, and dropout of 0.1. These were selected after hyperparameter tuning for this setting.

We use a Vision Transformer trunk with an embedding dimension of 256, four attention heads, and MLP dimension of 1024, and six layers. These were selected to achieve an architecture that trains quickly yet gets fairly good performance.

For all experiments, our training and evaluation is done on a single Nvidia RTX8000 GPU.

3.1 Patch Embedding

With this encoder we are able to achieve an accuracy of 85% on the validation using a patch size of 4. We also try patch sizes of 2, 8, and 16. All of these decrease performance. A patch size of 2 gets similar performance but runs 410% slower. A patch size of 8 runs in 71% and patch size of 16 in 57% of the time, but with much worse performance. See Figure 4 for performance comparison.

3.2 Initial Convolutions

We first evaluate different configurations of only a single convolution. We try out different kernel sizes, strides, and dilations.

Our best single-layer configuration achieves 87% accuracy (+2 from patch embedding), has kernel=5, stride=3, dilate=1, has 2% more parameters, and runs 43% slower. Using a dilation of 2 instead of 1 is twice as fast, but dramatically reduces performance. Using a stride of 2 instead of 3 runs half as

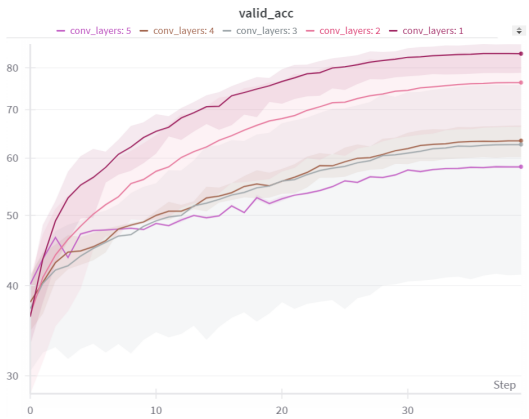


Figure 5: Adding additional convolutional layers hurts performance. Error bars show the performance spread over different values of kernel sizes and stride.

quickly, with only a small performance improvement. Increasing stride to 4 dramatically reduces performance, though it does run faster. A larger kernel of 7 instead of 5 runs a bit faster overall because it results in a slightly shorter sequence length for the transformer, but it doesn't perform as well.

Adding multiple layers of convolutions hurt performance, though adding additional layers does speed up performance. Results are shown in Figure 5. The best performing 2-layer-conv approach scored 2-3% lower than the best 1-layer-conv and had convolutions with kernel size 3 and stride 2 and runs 10% faster than patchifying but gets 1% worse performance. The best 3-layer-conv approach scored 11-12% lower than 1-layer and had kernel size 2 and stride 2 and runs 42% faster than patchifying but again gets worse performance. The increased speed here comes from increased downsampling of the sequence length, which results in less attention for the transformer to do, to some degradation in performance. The two layer approach here is very promising given the speedup and continued exploration into its parameterization may yield increased performance. Unlike [Xiao et al., 2021], we do not see that additional layers increase performance – and unlike them, we do not even decrease the size of the transformer, as the additional layers actually reduce the FLOPs of the model (though they increase its parameter count).

3.3 Vector Quantization

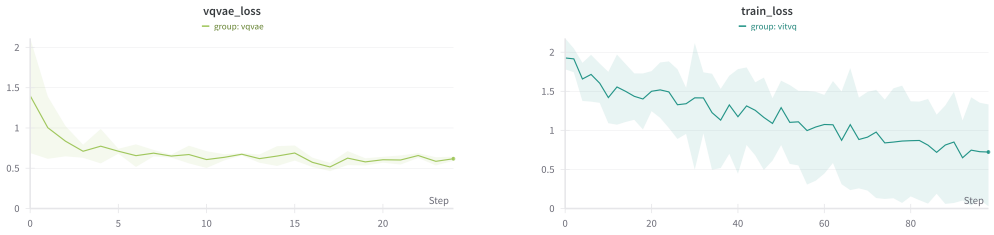
Our best Vector Quantization approaches hurt performance by nearly 10%, adds 12% more parameters, and runs 25-30% slower.

Since this approach can be applied after either Patch Embeddings or Convolutions, any slower choices in those settings stack with the slowdowns in this setting. Vector Quantization gets dramatically slower with large numbers of codes (we weren't able to go above a few hundred, though models like LLaMA use 32k tokens in the text-based setting and other language models can exceed 100k tokens). Our initial configuration of 640 codes in just one group reduced the model speed by 33x, so we weren't even able to train it.

In order to get better performance, we scale the number of groups (i.e. the number of distinct codebooks, each with separate codes). Our best configuration uses 64 codebooks with 32 codes each and a total embedding dimension of 256. Thus each code has 256 / 64 = 4 dimensions, and are concatenated with one another to form the output of the quantization step.

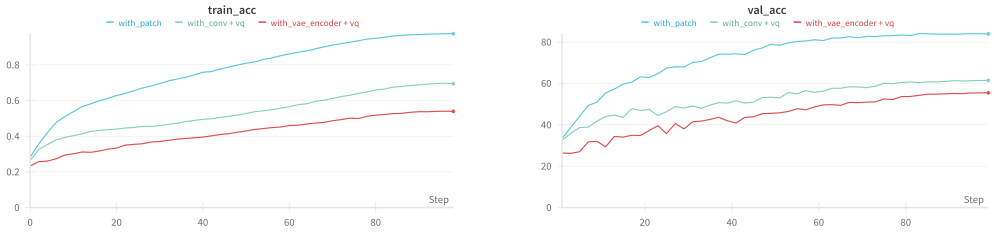
3.4 VQ-VAE Initialization

The VQ-VAE encoding scheme adds a huge number of parameters to the base vision transformer classifier, whilst also degrading performance to a significant level. The VQ-VAE begets a single large codebook with size of discrete latent space being 512 and each latent embedding has dimension of 64. It outputs encoding of size 4096, which is flattened to discrete codes of size $64 \times 8 \times 8$. These



(a) VQ-VAE training (b) ViT with VQ-VAE encoder training

Figure 6: Loss curves for training (a) VQ-VAE training on CIFAR-10 image reconstuction, (b) training VQ-VAE encoder applied to early ViT layers for CIFAR10 classification



(a) Training (b) Validation

Figure 7: (a) Training and (b) validation accuracy curves for the vq-vae experiments on CIFAR10. The larger encoder initial size before the ViT transformer blocks slows down the training process and eventually plateaus it at a much lower accuracy.

discrete representations from a trained codebook are used to classify using a ViT. Figure 6 shows the training loss curves for this set of experiments.

The VQ-VAE encoder (before vector quantization) exceeds the base ViT by 200% more parameters. In a variation of this application to control for the number of parameters in the encoding, a single convolution layer replaces the VQ-VAE encoder, and then the latents are discretized with the VQ layer, which improves the performance slightly over the VQ-VAE encoding but still under-performs a ViT trained with a patchify initialization by a large margin.

At the end of 50 epochs of CIFAR10 training, table 1 shows the performance and compute analysis for each method. Figure 7 shows the trends across epochs.

Table 1: VQ-VAE quantization experiments for 50 epochs

Encoding	Val Accuracy (%)	Params (M)	Training Time (s)
VQ-VAE encoder + VQ + ViT	55.47	13.5	3528
Conv layer + VQ + ViT	61.47	4.9	2590
ViT Patchify	84	4.8	905

This suggests that VQ-VAE discrete codes trained for image reconstruction cannot be leveraged for classification, and more task-specific training of an embedding space is the only way to leverage the plug-and-play idea for using a trained image latent space and discretization method.

3.5 Mixed-Resolution Tokenization

Mixed Resolution Tokenizer, replaces the `PatchEmbedding` layer in the baseline ViT, keeping the parameter count for the subsequent layers (`Transformer` block) exactly the same. It has a 12%

larger training time per epoch. This is due to the the forward passes through the saliency scorer to calculate $Score_{feat}[p]$ for each patch p.

We trained the baseline ViT, and Mixed-Res ViT to 30 epochs for CIFAR10 & CIFAR100 datasets.

At the end of these 30 epochs, the Mixed-Res ViT (80.40% for CIFAR10, 52.84% for CIFAR100) has a higher validation accuracy than the baseline ViT (74.79% for CIFAR10, 44.75% for CIFAR100). These results are provided in the graphs in Figure 8.

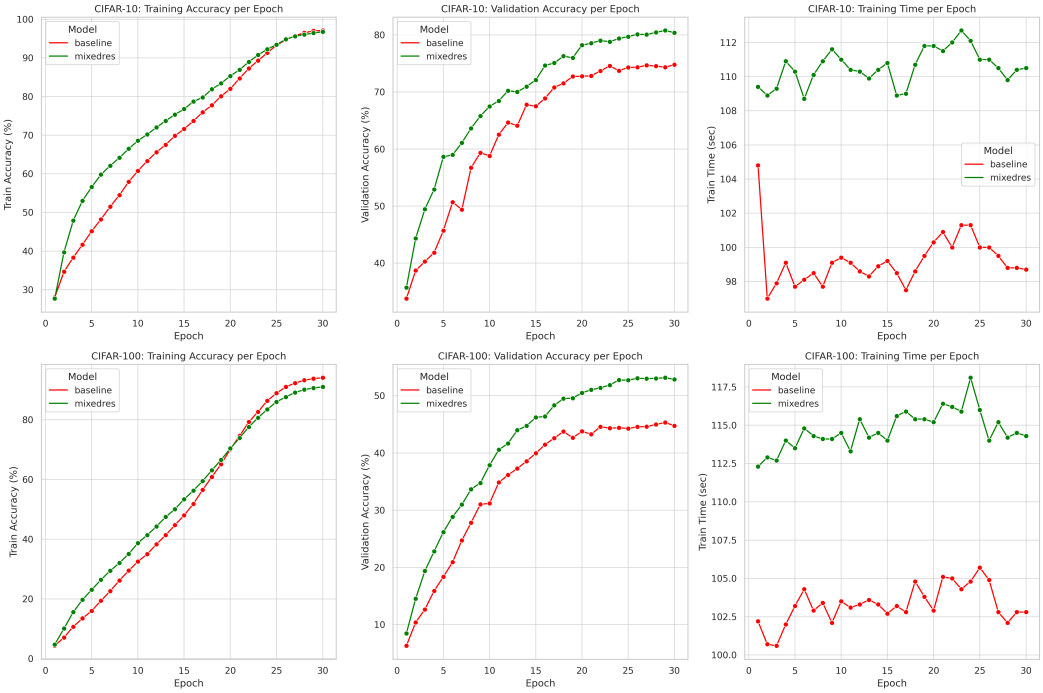


Figure 8: Training Accuracy, Validation Accuracy, Training Time per Epoch compared for a baseline ViT, and the same ViT with a Mixed Resolution Tokenizer replacing the PatchEmbedding layer.

4 Discussion

4.1 Limitations

Vector Quantize methods are well motivated due to their success in image reconstruction tasks, but they fail to adapt to classification tasks.

Convolutional networks help improve the performance of the ViT in early layers, but depending on the dataset and the size of the convolutional encoder (such as VQ-VAE encoder), it may worsen performance and slow down the training.

Mixed-Resolution Tokenizer is distribution dependent. CIFAR10/100 has similar images as the dataset on which the saliency scorer (ShuffleNet) is pre-trained: ImageNet. Applying this saliency scorer to images from a different distribution, say medical images, instead of CIFAR10 might not work.

4.2 Related and Future Work

Adaptive image tokenization methods such as employing a token learning module Ryoo et al. [2021] using convolution and attention layers has been shown to perform well for video ViT in reducing the depth of transformer layers and compute required for video recognition, and is a simple extension of the idea of using convolutional layers in the early ViT stages. Other methods like learning multiple adaptive codebooks instead of a single large codebook such as in VQ-VAE have previosuly been

used for image restoration and also show promise Liu et al. [2023] as a potential direction to test the trained latent space for classification. The marginal improvements in the mixed-resolution tokenizing scheme, and the superior performance of the patchify method combined suggest that modifying the patchify layer to compute adaptively learnt tokens for an efficient patchify module instead of a grid based one would be a promising next step. The aforementioned methods have not however been tested by their authors for simple image classification tasks, hence it forms a convincing basis for extension of this survey.

Methods like VQ-GAN Esser et al. [2021] are an extension of the VQ-VAE encoding idea, in which a GAN discriminator is used to discriminate the reconstructed images from a VQ-VAE and a GAN loss is backpropagated in addition to the VQ and reconstruction losses. This method outperforms VQ-VAE in learning an efficient discretized codebook for use in image reconstruction and infilling, but based on the results of our VQ-VAE experiments which adds a heavy parameter count to the base ViT classifier and slows down training, the VQ-GAN idea will likely produce similar results—since its latent space has been trained for image reconstruction tasks and not classification—with its rate of forward data and backward gradient flow is severely slowed down for image classification with a ViT.

5 Conclusion

After our review of different options for featurizing an image before feeding it into the convolutional network, we conclude that Patch Embeddings are a surprisingly robust and efficient approach. They require few parameters, are computational cheap, and provide robust performance on downstream tasks. Several layers of convolutions are a strong alternative; while they add many parameters to the model, they downsample the sequence length which reduces the amount of expensive attention steps that the transformer has to do and thus drastically speed up the model. Vector quantization had potential for helping the model to learn more robust concepts but practically did not improve performance and had significant computational costs. Vector quantize methods for a trained image encoder-decoder network do not extend themselves to task-agnostic performance for reusing the codebooks. Tokenizing schemes such as mixed resolution tokenizer reduces the number of input pixels and demonstrates slight improvement over the baseline, suggesting that more experiments in this direction of modifying the patchifying module would yield promising results in terms of improving classification accuracy at reduced computational costs and training time.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis, 2021.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009. URL `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

Kechun Liu, Yitong Jiang, Inchang Choi, and Jinwei Gu. Learning image-adaptive codebooks for class-agnostic image restoration, 2023.

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design, 2018.

Tomer Ronen, Omer Levy, and Avram Golbert. Vision transformers with mixed-resolution tokenization, 2023.

Michael Ryoo, AJ Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. Tokenlearner: Adaptive space-time tokenization for videos. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 12786–12797. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/6a30e32e56fce5cf381895dfe6ca7b6f-Paper.pdf.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units, 2016.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

Changhan Wang, Kyunghyun Cho, and Jiatao Gu. Neural machine translation with byte-level subwords, 2019.

Tete Xiao, Mannat Singh, Eric Mintun, Trevor Darrell, Piotr Dollár, and Ross Girshick. Early convolutions help transformers see better, 2021.

Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. Megabyte: Predicting million-byte sequences with multiscale transformers, 2023.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification, 2016.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices, 2017.