

Exercise 11 Part 2: Vision Transformers

Summer Semester 2024

Author: Stefan Baumann (stefan.baumann@lmu.de)

Task: Implement & Train a ViT

Refer to the lecture and the original ViT paper (*AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE*, Dosovitskiy et al., 2020) for details. The naming of the hyperparameters is as in the aforementioned paper.

Similar to Part 1, you're expected to implement each block yourself, although you're allowed to use blocks like `torch.nn.MultiheadAttention`, `torch.nn.Linear`, etc. Implement the blocks as in the original ViT paper. No usage of things such as full pre-made FFN/self-attention blocks or full transformer implementations like `torchvision.models.vision_transformer.VisionTransformer` is allowed for this exercise. You're expected to do full vectorized implementations in native PyTorch (again, einops is allowed) without relying on Python for loops for things such as patching etc.

Some relevant details:

- For simplicity of implementation, we will use a randomly (Gaussian with mean 0 and variance 1) initialized *learnable* positional embedding, not a Fourier/sinusoidal one.
- Don't forget about all of the layer norms!
- Consider the `batch_first` attribute of `nn.MultiheadAttention`, should you use that class
- We'll make the standard assumption that $dim_{\text{head}} = dim_{\text{hidden}} / N_{\text{heads}}$

```
import math

import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.transforms as T
from torchvision.datasets import CIFAR10
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
from tqdm.auto import tqdm

# Optional
import einops

device = 'mps' if torch.backends.mps.is_available() else ('cuda' if
torch.cuda.is_available() else 'cpu')
print(f'Using device "{device}"')

Using device "mps".
```

```
/Users/janinaalicamattes/miniforge3/envs/pytorch-pyl1/lib/python3.11/
site-packages/tqdm/auto.py:21: TqdmWarning: IPProgress not found.
Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
from .autonotebook import tqdm as notebook_tqdm
```

```
class ResidualModule(nn.Module):
    def __init__(
        self,
        inner_module: nn.Module
    ):
        super().__init__()
        self.inner_module = inner_module

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return x + self.inner_module(x)

class FeedForwardBlock(nn.Module):
    """ FeedForwardBlock class for the MLP of a transformer block in
    the Transformer Encoder.
        The linear MLP layers are local and translationally
    equivariant,
        while the self-attention layers are global and permutation
    invariant.

        Args:
            hidden_size (int): Hidden size of the model.
            mlp_size (int): Size of the MLP.
            p_dropout (float): Dropout probability.

        Returns:
            torch.Tensor: Output tensor of the feedforward block.
    """
    def __init__(
        self,
        hidden_size: int,
        mlp_size: int,
        p_dropout: float
    ):
        super().__init__()
        self.dropout = p_dropout
        self.hidden_size = hidden_size # kept fixed
        self.mlp_size = mlp_size

        self.linear1 = nn.Linear(self.hidden_size, self.mlp_size)
        self.dropout1 = nn.Dropout(self.dropout)
        self.gelu = nn.GELU()
        self.linear2 = nn.Linear(self.mlp_size, self.hidden_size)
        self.dropout2 = nn.Dropout(self.dropout)
```

```

def forward(self, x: torch.Tensor) -> torch.Tensor:
    x = self.linear1(x)
    x = self.gelu(x)
    x = self.dropout1(x)
    x = self.linear2(x)
    x = self.dropout2(x)
    return x

class SelfAttentionTransformerBlock(nn.Module):
    """ SelfAttentionTransformerBlock class for the transformer block
    in the Transformer Encoder.
    The linear MLP layers are local and translationally
    equivariant,
    while the self-attention layers are global and permutation
    invariant.

    Args:
        hidden_size (int): Hidden size of the model.
        n_heads (int): Number of heads in the multi-head self-
attention.
        p_dropout (float): Dropout probability.

    Returns:
        torch.Tensor: Output tensor of the transformer block.
    """

    def __init__(
        self,
        hidden_size: int,
        n_heads: int,
        p_dropout: float
    ):
        super().__init__()
        self.hidden_size = hidden_size
        self.n_heads = n_heads
        self.p_dropout = p_dropout
        self.mlp_size = self.hidden_size * 4 # Standard in the
literature

        # Layer normalization
        self.norm1 = nn.LayerNorm(self.hidden_size)
        self.norm2 = nn.LayerNorm(self.hidden_size)

        # Multi-head self-attention
        self.mha = nn.MultiheadAttention(self.hidden_size,
self.n_heads, dropout=p_dropout, batch_first=True)

        # MLP block
        self.mlp = FeedForwardBlock(self.hidden_size, self.mlp_size,

```

```

self.p_dropout)

    # Dropout
    self.dropout = nn.Dropout(self.p_dropout)

    def forward(self, x: torch.Tensor) -> torch.Tensor:

        # Residual connection
        residual = x

        # Self-attention (global and permutation invariant)
        x = self.norm1(x)
        x = self.mha(x, x, x)[0]
        x = self.dropout(x)
        # Residual connection
        x = x + residual

        # Residual connection
        residual = x

        # MLP (local and translationally equivariant)
        x = self.norm2(x)
        x = self.mlp(x)
        x = self.dropout(x)
        # Residual connection
        x = x + residual

        return x

class VisionTransformer(nn.Module):
    def __init__(
        self,
        in_channels: int = 3,
        patch_size: int = 4,
        image_size: int = 32,
        layers: int = 6,
        hidden_size: int = 256,
        mlp_size: int = 512,
        n_heads: int = 8,
        num_classes: int = 10,
        p_dropout: float = 0.2,
    ):
        super().__init__()

        self.in_channels = in_channels
        self.patch_size = patch_size
        self.image_size = image_size
        self.layers = layers
        self.hidden_size = hidden_size

```

```

self.mlp_size = mlp_size
self.n_heads = n_heads
self.num_classes = num_classes
self.p_dropout = p_dropout

# -----
# ----- Transformer Encoder -----
# -----

# Image patches / token
self.num_patches = (self.image_size // self.patch_size) ** 2 #
Number of patches (L) or tokens
self.patch_dim = self.in_channels * (self.patch_size ** 2) #
Dimension of the patch after flattening (D)

# Patch embedding - linear projection of the patches
self.patch_embed = nn.Linear(self.patch_dim, self.hidden_size)

# Positional encoding - learnable positional embeddings
self.pos_embed = nn.Parameter(torch.randn(1, self.num_patches
+ 1, hidden_size))

# CLS token - classification token
self.cls_token = nn.Parameter(torch.zeros(1, 1,
self.hidden_size))

# Transformer blocks
self.transformer_blocks = nn.Sequential(*[
    SelfAttentionTransformerBlock(self.hidden_size,
self.n_heads, self.p_dropout)
    for _ in range(self.layers)
])

# Dropout
self.dropout = nn.Dropout(self.p_dropout)

# -----
# ----- Classification head -----
# -----

self.norm = nn.LayerNorm(self.hidden_size)
self.classification_head = nn.Linear(self.hidden_size,
self.num_classes)

# Initialize weights
self._init_weights()

def _init_weights(self):

```

```

    # Initialize weights
    self.apply(self._init_layer_weights)

def _init_layer_weights(self, m):
    # Initialize weights of the model
    if isinstance(m, nn.Linear):
        nn.init.xavier_uniform_(m.weight)
        if m.bias is not None:
            nn.init.zeros_(m.bias)
    elif isinstance(m, nn.LayerNorm):
        nn.init.ones_(m.weight)
        nn.init.zeros_(m.bias)

def patchify(self, x: torch.Tensor) -> torch.Tensor:
    """Takes an image tensor of shape (B, C, H, W) and transforms
    it to a sequence of patches (B, L, D), with a learnable linear
    projection after flattening,
    and a standard additive positional encoding applied. Note that
    the activations in (Vision) Transformer implementations are
    typically passed around in channels-last layout, different
    from typical PyTorch norms.

    The linear projection of flattened image patches produces
    lower-dimensional linear embeddings from flattened patches and adds
    positional embeddings.

    Args:
        x (torch.Tensor): Input tensor of shape (B, C, H, W)

    Returns:
        torch.Tensor: Embedded patch sequence tensor with
        positional encodings applied and shape (B, L, D)
    """
    B, C, H, W = x.shape

    # Reshape and flatten the image patches
    x = x.reshape(B, C, H // self.patch_size, self.patch_size, W
    // self.patch_size, self.patch_size)
    x = x.permute(0, 2, 4, 1, 3, 5).contiguous() #
    Size: (B, H, W, C, patch_size, patch_size)
    x = x.view(B, -1, C * self.patch_size * self.patch_size) #
    Size: (B, L, D) with D: C * patch_size * patch_size

    # Linear projection of the patches
    x = self.patch_embed(x)

    # Add positional embeddings
    x = x + self.pos_embed[:, 1:, :]

```

```

    # Add CLS token
    cls_token = self.cls_token.expand(B, -1, -1)
    x = torch.cat((cls_token, x), dim=1)

    # Add positional embeddings for the CLS token
    x[:, 0, :] = x[:, 0, :] + self.pos_embed[:, 0, :]

    return x

def forward(self, x: torch.Tensor) -> torch.Tensor:
    """Takes an image tensor of shape (B, C, H, W), applies
    patching, a standard ViT
    and then an output projection of the CLS token
    to finally create a class logit prediction of shape (B,
    N_cls)

    Args:
        x (torch.Tensor): Input tensor of shape (B, C, H, W)

    Returns:
        torch.Tensor: Output logits of shape (B, N_cls)
    """
    # Patchify input image + pos embeddings
    x = self.patchify(x)

    # Apply dropout
    x = self.dropout(x)

    # Transformer blocks
    x = self.transformer_blocks(x)

    # Classification head
    x = self.norm(x[:, 0]) # select only the learned
    # CLS token with Size: (B, D)
    x = self.classification_head(x)

    return x

```

Training

Do not modify this code! You are free to modify the four parameters in the first block, although no modifications should be necessary to achieve >70% validation accuracy with a correct transformer implementation.

```

DATASET_CACHE_DIR = './data'
BATCH_SIZE = 128
LR = 3e-4
N_EPOCHS = 50

```

```

transforms_val = T.Compose([
    T.ToTensor(),
    T.Normalize([0.49139968, 0.48215841, 0.44653091], [0.24703223,
0.24348513, 0.26158784])),
])
transforms_train = T.Compose([
    T.RandomHorizontalFlip(),
    T.RandomResizedCrop((32, 32), scale=(0.8, 1.0), ratio=(0.9, 1.1)),
    T.ToTensor(),
    T.Normalize([0.49139968, 0.48215841, 0.44653091], [0.24703223,
0.24348513, 0.26158784])),
])

model = VisionTransformer().to(device)
optim = torch.optim.Adam(model.parameters(), lr=LR)
loss_fn = nn.CrossEntropyLoss()

dataloader_train = DataLoader(CIFAR10(root=DATASET_CACHE_DIR,
train=True, download=True, transform=transforms_train),
batch_size=BATCH_SIZE, shuffle=True, drop_last=True, num_workers=4)
dataloader_val = DataLoader(CIFAR10(root=DATASET_CACHE_DIR,
train=False, download=True, transform=transforms_val),
batch_size=BATCH_SIZE, shuffle=False, drop_last=False, num_workers=4)

train_losses = []
val_accs = []

for i_epoch in range(N_EPOCHS):
    for i_step, (images, labels) in (pbar :=
tqdm(enumerate(dataloader_train), desc=f'Training (Epoch {i_epoch +
1}/{N_EPOCHS}'))):
        optim.zero_grad()
        loss = loss_fn(model(images.to(device)), labels.to(device))
        loss.backward()
        optim.step()

        # Some logging
        loss_val = loss.detach().item()
        train_losses.append(loss_val)
        pbar.set_postfix({ 'loss': loss_val } | ({ 'val_acc':
val_accs[-1] } if len(val_accs) > 0 else { }))

    # Validation every epoch
    with torch.no_grad():
        n_total, n_correct = 0, 0
        for i_step, (images, labels) in (pbar :=
tqdm(enumerate(dataloader_val), desc='Validating')):
            predicted = model(images.to(device)).argmax(dim=-1)
            n_correct += (predicted.cpu() ==
labels).float().sum().item())

```



```
        n_total += labels.shape[0]
    val_accs.append(n_correct / n_total)
    print(f'Validation accuracy: {val_accs[-1]:.3f}')
```

```
plt.figure(figsize=(6, 3))
plt.subplot(121)
plt.plot(train_losses)
plt.xlabel('Steps')
plt.ylabel('Training Loss')
plt.subplot(122)
plt.plot(val_accs)
plt.xlabel('Epochs')
plt.ylabel('Validation Accuracy')
plt.tight_layout()
plt.show()
```

Downloading <https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz> to
./data/cifar-10-python.tar.gz

100%|██████████| 170498071/170498071 [00:03<00:00, 48744481.98it/s]

Extracting ./data/cifar-10-python.tar.gz to ./data

```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/
dataloader.py:558: UserWarning: This DataLoader will create 4 worker
processes in total. Our suggested max number of worker in current
system is 2, which is smaller than what this DataLoader is going to
create. Please be aware that excessive worker creation might get
DataLoader running slow or even freeze, lower the worker number to
avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
```

Files already downloaded and verified

```
{"model_id": "3e6faeb34bf948ebb159c027f333703f", "version_major": 2, "vers
ion_minor": 0}
```

```
{"model_id": "3882826ed0184bf08f9a8c0afba3f140", "version_major": 2, "vers
ion_minor": 0}
```

Validation accuracy: 0.419

```
{"model_id": "0ed484f339c941d28836d7cdc8270e55", "version_major": 2, "vers
ion_minor": 0}
```

```
{"model_id": "5f348dd431414186a8173d8e2043e0a3", "version_major": 2, "vers
ion_minor": 0}
```

Validation accuracy: 0.469

```
{"model_id": "b8f28aa5e16442b59f2a9c24b2e90a4a", "version_major": 2, "vers
ion_minor": 0}
```

```
{"model_id": "9ae3787346ac4c3bb2c1ddb5d0b1e4ef", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.519

```
{"model_id": "2c875ebce3bc44ee8cc36b8abecb783a", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "a49f78e88d624e4dbd0f8ad6b2fbd928", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.545

```
{"model_id": "89a5080fde974a3b97aa4c9f12effaed", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "242a2a96ae33450aaea9d7d16751bb6c", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.541

```
{"model_id": "b4b8f1c5ed9449bba382d3e4f35f23c4", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "3c131ee81818444182dcb946ad66a5d4", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.566

```
{"model_id": "5cd59c6816d2466d8fce80501c9e9f93", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "a547e48da3ac4d70838d41397a78f266", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.562

```
{"model_id": "e1fb90e089eb4f0a97913a2055da35c6", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "74e71b27828c4cbf9bb4cd111bce7a78", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.596

```
{"model_id": "60e6093a63284c0d85a9789f3a742a46", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "fb5c2b24b80540e89339d66337770615", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.600

```
{"model_id": "e6d8cf12c51c4c699a46153059fa02ea", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "6e573e0a9df44f7bbb1abae8b040e3c7", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.598

```
{"model_id": "57fa4238d85a4c9b80f64e5bd837fc56", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "774b699d70da4624b19de731da677fe9", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.610

```
{"model_id": "ed331391005c40b19e99cc4ba586e722", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "abc111df66b049ebbef194ba5a18fccc", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.618

```
{"model_id": "bbb3ca2688e340f8807fce4f6125653e", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "cc1c464074d04f99ad8dacd167d00ec6", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.614

```
{"model_id": "160bab65fe514a3e95d0ebd219d238e7", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "9f80a9156ee8409791bc608ad5d77213", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.629

```
{"model_id": "642a3f9e23214d4d8608533835f4cf59", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "a78743ec23dc49d3915932060b73521f", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.636

```
{"model_id": "abd33fea110d4fdda3c3d76b55a7e8ad", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "f32db9f5c9e44c548e4bdbc7774e4386", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.645

```
{"model_id": "1480a51e53af433a957ad04313578b7c", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "d9c210e82a8d45ed8bd3bf3b685e6e3f", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.652

```
{"model_id": "048afe2661f54639895f565a3fb4da33", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "fd02315ddd454ac7a3260e9d2f226927", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.646

```
{"model_id": "528705e1b99d4124a2b009a771d7bd31", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "dc3c3894cb464cf49caee3f8985ab203", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.648

```
{"model_id": "4eda519083a64fcfad8d58785bb4bd41", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "c92429c7b51d4587ad924f69c3d15afc", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.659

```
{"model_id": "1749b80d0f344bb78fb49f4c54be7b5e", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "62f9a38640a34b40ade5c8cac63d27ae", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.653

```
{"model_id": "4f425b8c002d46eda31b897de85682d0", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "d193b0c95ae348cf953f1756d4f648dd", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.664

```
{"model_id": "1732b59c84ec4ebaabfeedbc614bee54", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "beff4e308da24b03997ef7f22da1ad29", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.669

```
{"model_id": "2e3bf9a7f5c0413d86b0bdca235b0a45", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "9599f19f7b69496589cf27189d1936e0", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.666

```
{"model_id": "43dae5a791b8497b9da51374ee37209e", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "d1f057c066364f9e98a4c742c3b5b1c8", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.672

```
{"model_id": "2d490ed1e6ad4a07bc0e1d7fa7b57fb4", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "2fa5d5a1b34048e597c0280682267ab7", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.678

```
{"model_id": "2758917c1f5e4c89b4dd2dc9b3df0dbe", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "e3d884ba52ad49ebb0401670faf64f77", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.673

```
{"model_id": "72fb76d71bf24f19a787de5a2eeff7ac", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "ee5e53e745f14f2bad2cc1768bff16a7", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.678

```
{"model_id": "2cf8ea64cd104c679fd24ccd2f210102", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "bafdl1a32883c4b25b7b46229f7775ac9", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.678

```
{"model_id": "428327532b1742118646e2901d8afb9d", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "30eaa8d56e4a4f1fb1eb1125b818d41e", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.692

```
{"model_id": "70f17f255f844305be48a65e654a34c2", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "3d3a0775791c4d54a7350c8cae75ca86", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.690

```
{"model_id": "a82464b0b3a243749edfa30a046ffa53", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "da56ea67dbfc452087403e8ee9297334", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.690

```
{"model_id": "b6e6a409f15e4d81a37753e05a247c6e", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "2b89ee5e060b49999c720291a965201a", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.699

```
{"model_id": "0e282444e62649859715a3610a53e5a4", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "96159cba37814233adc5eb557cc62813", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.697

```
{"model_id": "5b1b17f73404427595664ed761b6383e", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "c1b0ae31c12d45ebb063fd17657b07de", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.697

```
{"model_id": "cc00bfe3f9254732881251e665b9b99e", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "7d7d5ecfe07747f7b6b707ce4141a043", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.706

```
{"model_id": "375d75c0e2f04426a62247fc93b2a604", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "d58d52c568ed482aba1cf1601e625b51", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.707

```
{"model_id": "e49ce9bd8acb4f6e9eba6be4b426cf9a", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "cc2b1ce47cf849c6b12c87b5d170a887", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.716

```
{"model_id": "84b648ec7ba44638906c144fc8a4e19b", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "213b388b5d9c40aab2bb979ebcb0b4c2", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.712

```
{"model_id": "0c60e841b6ea422d91e192bb0e89af9a", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "364524fe04c4436f8299e48a5b646def", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.712

```
{"model_id": "ff71350dfb2741229cce070242963d5c", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "af15b4906c22482b8eb262f8ad9e3aaf", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.720

```
{"model_id": "ab20c40499094c6eadf21c2165f440fd", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "4e599fd4fe27427e80c62f2fa6dd9c5b", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.717

```
{"model_id": "45c9425293f14f628dfcd9b147a49653", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "181a615f97c640f1adf096167eecf3d3", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.719

```
{"model_id": "e3bb4caeeb7a47d08b19f7bac8825e00", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "39f8ec579bdb485fb64bad8d2e64db9a", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.714

```
{"model_id": "207b6cbafc744deae2613bc1771d436", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "d5cef29d0cd1476993fc5268ce32feb9", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.720

```
{"model_id": "109234e4b7c441208c42aef1fde01339", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "4f2b884fc96f45feaf5d3265982dcca3", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.728

```
{"model_id": "8a45ea57a4024eec8d65faf518791b44", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "9a6b20ec5d354daba722437118ca1edf", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.724

```
{"model_id": "b6d2938d210a440abf2d02fb468252b9", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "6960d9d2f9964ff3a444652f892652e9", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.721

```
{"model_id": "0f464478fda440cf92031fcd924820ce", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "ebc2ef898cf3432fa45a04e80b590cb0", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.728


```
{"model_id": "100d2ebc019a42f9b1d3c5f456f76312", "version_major": 2, "version_minor": 0}
```

```
{"model_id": "1cc8aa590434473ebf88b9f462e148ed", "version_major": 2, "version_minor": 0}
```

Validation accuracy: 0.725

