Keras

Star 55,494

About Keras
Getting started
Developer guides
Keras API reference
Code examples
Computer Vision
Natural Language Processing
Structured Data
Timeseries
Audio Data
Generative Deep Learning
Reinforcement Learning
Graph Data
Quick Keras Recipes
Why choose Keras?
Community & governance
Contributing to Keras
KerasTuner
KerasCV
KerasNLP

Search Keras documentation...

» Code examples / Computer Vision / Image classification with Vision Transformer

Image classification with Vision Transformer

Author: Khalid Salama
Date created: 2021/01/18
Last modified: 2021/01/18
Description: Implementing the Vision Transformer (ViT) model for image classification.

View in Colab • GitHub source

Introduction

This example implements the Vision Transformer (ViT) model by Alexey Dosovitskiy et al. for image classification, and demonstrates it on the CIFAR-100 dataset. The ViT model applies the Transformer architecture with self-attention to sequences of image patches, without using convolution layers.

This example requires TensorFlow 2.4 or higher, as well as TensorFlow Addons, which can be installed using the following command:

```
pip install -U tensorflow-addons
```

Setup

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_addons as tfa
```

Prepare the data

```
num_classes = 100
input_shape = (32, 32, 3)

(x_train, y_train), (x_test, y_test) = keras.datasets.cifar100.load_data()

print(f"x_train shape: {x_train.shape} - y_train shape: {y_train.shape}")
print(f"x_test shape: {x_test.shape} - y_test shape: {y_test.shape}")
```

```
x_train shape: (50000, 32, 32, 3) - y_train shape: (50000, 1)
x_test shape: (10000, 32, 32, 3) - y_test shape: (10000, 1)
```

Configure the hyperparameters

```
learning_rate = 0.001
weight_decay = 0.0001
batch_size = 256
num_epochs = 100
image_size = 72  # We'll resize input images to this size
patch_size = 6  # Size of the patches to be extract from the input images
num_patches = (image_size // patch_size) ** 2
projection_dim = 64
num_heads = 4
transformer_units = [
    projection_dim * 2,
    projection_dim,
]  # Size of the transformer layers
transformer_layers = 8
mlp_head_units = [2048, 1024]  # Size of the dense layers of the final classifier
```

Use data augmentation

```
data_augmentation = keras.Sequential(
    [
        layers.Normalization(),
        layers.Resizing(image_size, image_size),
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(factor=0.02),
        layers.RandomZoom(
            height_factor=0.2, width_factor=0.2
        ),
    ],
    name="data_augmentation",
)
# Compute the mean and the variance of the training data for normalization.
data_augmentation.layers[0].adapt(x_train)
```

Implement multilayer perceptron (MLP)

```
def mlp(x, hidden_units, dropout_rate):
    for units in hidden_units:
        x = layers.Dense(units, activation=tf.nn.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x
```

Implement patch creation as a layer

```
class Patches(layers.Layer):
    def __init__(self, patch_size):
        super(Patches, self).__init__()
        self.patch_size = patch_size

    def call(self, images):
        batch_size = tf.shape(images)[0]
        patches = tf.image.extract_patches(
            images=images,
            sizes=[1, self.patch_size, self.patch_size, 1],
            strides=[1, self.patch_size, self.patch_size, 1],
            rates=[1, 1, 1, 1],
            padding="VALID",
        )
        patch_dims = patches.shape[-1]
        patches = tf.reshape(patches, [batch_size, -1, patch_dims])
        return patches
```

Let's display patches for a sample image

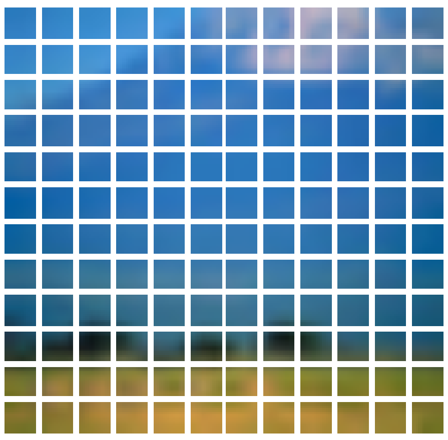
```
import matplotlib.pyplot as plt

plt.figure(figsize=(4, 4))
image = x_train[np.random.choice(range(x_train.shape[0]))]
plt.imshow(image.astype("uint8"))
plt.axis("off")

resized_image = tf.image.resize(
    tf.convert_to_tensor([image]), size=(image_size, image_size)
)
patches = Patches(patch_size)(resized_image)
print(f"Image size: {image_size} X {image_size}")
print(f"Patch size: {patch_size} X {patch_size}")
print(f"Patches per image: {patches.shape[1]}")
print(f"Elements per patch: {patches.shape[-1]}")

n = int(np.sqrt(patches.shape[1]))
plt.figure(figsize=(4, 4))
for i, patch in enumerate(patches[0]):
    ax = plt.subplot(n, n, i + 1)
    patch_img = tf.reshape(patch, (patch_size, patch_size, 3))
    plt.imshow(patch_img.numpy().astype("uint8"))
    plt.axis("off")
```

```
Image size: 72 X 72
Patch size: 6 X 6
Patches per image: 144
Elements per patch: 108
```



Implement the patch encoding layer

The `PatchEncoder` layer will linearly transform a patch by projecting it into a vector of size `projection_dim`. In addition, it adds a learnable position embedding to the projected vector.

```
class PatchEncoder(layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super(PatchEncoder, self).__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(units=projection_dim)
        self.position_embedding = layers.Embedding(
            input_dim=num_patches, output_dim=projection_dim
        )

    def call(self, patch):
        positions = tf.range(start=0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch) + self.position_embedding(positions)
        return encoded
```

Build the ViT model

The ViT model consists of multiple Transformer blocks, which use the `layers.MultiHeadAttention` layer as a self-attention mechanism applied to the sequence of patches. The Transformer blocks produce a `[batch_size, num_patches, projection_dim]` tensor, which is processed via an classifier head with softmax to produce the final class probabilities output.

Unlike the technique described in the paper, which prepends a learnable embedding to the sequence of encoded patches to serve as the image representation, all the outputs of the final Transformer block are reshaped with `layers.Flatten()` and used as the image representation input to the classifier head. Note that the `layers.GlobalAveragePooling1D` layer could also be used instead to aggregate the outputs of the Transformer block, especially when the number of patches and the projection dimensions are large.

```
def create_vit_classifier():
    inputs = layers.Input(shape=input_shape)
    # Augment data.
    augmented = data_augmentation(inputs)
    # Create patches.
    patches = Patches(patch_size)(augmented)
    # Encode patches.
    encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)

    # Create multiple layers of the Transformer block.
    for _ in range(transformer_layers):
        # Layer normalization 1.
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        # Create a multi-head attention layer.
        attention_output = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=projection_dim, dropout=0.1
        )(x1, x1)
        # Skip connection 1.
        x2 = layers.Add()([attention_output, encoded_patches])
        # Layer normalization 2.
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        # MLP.
        x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)
        # Skip connection 2.
        encoded_patches = layers.Add()([x3, x2])

    # Create a [batch_size, projection_dim] tensor.
    representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.5)(representation)
    # Add MLP.
    features = mlp(representation, hidden_units=mlp_head_units, dropout_rate=0.5)
    # Classify outputs.
    logits = layers.Dense(num_classes)(features)
    # Create the Keras model.
    model = keras.Model(inputs=inputs, outputs=logits)
    return model
```

Compile, train, and evaluate the mode

```
def run_experiment(model):
    optimizer = tfa.optimizers.AdamW(
        learning_rate=learning_rate, weight_decay=weight_decay
    )

    model.compile(
        optimizer=optimizer,
        loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=[
            keras.metrics.SparseCategoricalAccuracy(name="accuracy"),
            keras.metrics.SparseTopKCategoricalAccuracy(5, name="top-5-accuracy"),
        ],
    )

    checkpoint_filepath = "/tmp/checkpoint"
    checkpoint_callback = keras.callbacks.ModelCheckpoint(
        checkpoint_filepath,
        monitor="val_accuracy",
        save_best_only=True,
        save_weights_only=True,
    )

    history = model.fit(
        x=x_train,
        y=y_train,
        batch_size=batch_size,
        epochs=num_epochs,
        validation_split=0.1,
        callbacks=[checkpoint_callback],
    )

    model.load_weights(checkpoint_filepath)
    _, accuracy, top_5_accuracy = model.evaluate(x_test, y_test)
    print(f"Test accuracy: {round(accuracy * 100, 2)}%")
    print(f"Test top 5 accuracy: {round(top_5_accuracy * 100, 2)}%")

    return history


vit_classifier = create_vit_classifier()
history = run_experiment(vit_classifier)
```

```
Epoch 1/100
176/176 [==============================] - 33s 136ms/step - loss: 4.8863 - accuracy: 0.0294 -
top-5-accuracy: 0.1117 - val_loss: 3.9661 - val_accuracy: 0.0992 - val_top-5-accuracy: 0.3056
Epoch 2/100
176/176 [==============================] - 22s 127ms/step - loss: 4.0162 - accuracy: 0.0865 -
top-5-accuracy: 0.2683 - val_loss: 3.5691 - val_accuracy: 0.1630 - val_top-5-accuracy: 0.4226
Epoch 3/100
176/176 [==============================] - 22s 127ms/step - loss: 3.7313 - accuracy: 0.1254 -
top-5-accuracy: 0.3535 - val_loss: 3.3455 - val_accuracy: 0.1976 - val_top-5-accuracy: 0.4756
Epoch 4/100
176/176 [==============================] - 23s 128ms/step - loss: 3.5411 - accuracy: 0.1541 -
top-5-accuracy: 0.4121 - val_loss: 3.1925 - val_accuracy: 0.2274 - val_top-5-accuracy: 0.5126
Epoch 5/100
176/176 [==============================] - 22s 127ms/step - loss: 3.3749 - accuracy: 0.1847 -
top-5-accuracy: 0.4572 - val_loss: 3.1043 - val_accuracy: 0.2388 - val_top-5-accuracy: 0.5320
Epoch 6/100
176/176 [==============================] - 22s 127ms/step - loss: 3.2589 - accuracy: 0.2057 -
top-5-accuracy: 0.4906 - val_loss: 2.9319 - val_accuracy: 0.2782 - val_top-5-accuracy: 0.5756
Epoch 7/100
176/176 [==============================] - 22s 127ms/step - loss: 3.1165 - accuracy: 0.2331 -
top-5-accuracy: 0.5273 - val_loss: 2.8072 - val_accuracy: 0.2972 - val_top-5-accuracy: 0.5946
Epoch 8/100
176/176 [==============================] - 22s 127ms/step - loss: 2.9902 - accuracy: 0.2563 -
top-5-accuracy: 0.5556 - val_loss: 2.7207 - val_accuracy: 0.3188 - val_top-5-accuracy: 0.6258
Epoch 9/100
176/176 [==============================] - 22s 127ms/step - loss: 2.8828 - accuracy: 0.2800 -
top-5-accuracy: 0.5827 - val_loss: 2.6396 - val_accuracy: 0.3244 - val_top-5-accuracy: 0.6402
Epoch 10/100
176/176 [==============================] - 23s 128ms/step - loss: 2.7824 - accuracy: 0.2997 -
top-5-accuracy: 0.6110 - val_loss: 2.5580 - val_accuracy: 0.3494 - val_top-5-accuracy: 0.6568
Epoch 11/100
176/176 [==============================] - 23s 130ms/step - loss: 2.6743 - accuracy: 0.3209 -
top-5-accuracy: 0.6333 - val_loss: 2.5000 - val_accuracy: 0.3594 - val_top-5-accuracy: 0.6726
Epoch 12/100
176/176 [==============================] - 23s 130ms/step - loss: 2.5800 - accuracy: 0.3431 -
top-5-accuracy: 0.6522 - val_loss: 2.3900 - val_accuracy: 0.3798 - val_top-5-accuracy: 0.6878
Epoch 13/100
176/176 [==============================] - 23s 128ms/step - loss: 2.5019 - accuracy: 0.3559 -
top-5-accuracy: 0.6671 - val_loss: 2.3464 - val_accuracy: 0.3960 - val_top-5-accuracy: 0.7002
Epoch 14/100
176/176 [==============================] - 22s 128ms/step - loss: 2.4207 - accuracy: 0.3728 -
top-5-accuracy: 0.6905 - val_loss: 2.3130 - val_accuracy: 0.4032 - val_top-5-accuracy: 0.7040
Epoch 15/100
176/176 [==============================] - 23s 128ms/step - loss: 2.3371 - accuracy: 0.3932 -
top-5-accuracy: 0.7093 - val_loss: 2.2447 - val_accuracy: 0.4136 - val_top-5-accuracy: 0.7202
Epoch 16/100
176/176 [==============================] - 23s 128ms/step - loss: 2.2650 - accuracy: 0.4077 -
top-5-accuracy: 0.7201 - val_loss: 2.2101 - val_accuracy: 0.4222 - val_top-5-accuracy: 0.7246
Epoch 17/100
176/176 [==============================] - 22s 127ms/step - loss: 2.1822 - accuracy: 0.4204 -
top-5-accuracy: 0.7376 - val_loss: 2.1446 - val_accuracy: 0.4344 - val_top-5-accuracy: 0.7416
Epoch 18/100
176/176 [==============================] - 22s 128ms/step - loss: 2.1485 - accuracy: 0.4284 -
top-5-accuracy: 0.7476 - val_loss: 2.1094 - val_accuracy: 0.4432 - val_top-5-accuracy: 0.7454
Epoch 19/100
176/176 [==============================] - 22s 128ms/step - loss: 2.0717 - accuracy: 0.4464 -
top-5-accuracy: 0.7618 - val_loss: 2.0718 - val_accuracy: 0.4584 - val_top-5-accuracy: 0.7570
Epoch 20/100
176/176 [==============================] - 22s 127ms/step - loss: 2.0031 - accuracy: 0.4605 -
top-5-accuracy: 0.7731 - val_loss: 2.0286 - val_accuracy: 0.4610 - val_top-5-accuracy: 0.7654
Epoch 21/100
176/176 [==============================] - 22s 127ms/step - loss: 1.9650 - accuracy: 0.4700 -
top-5-accuracy: 0.7820 - val_loss: 2.0225 - val_accuracy: 0.4642 - val_top-5-accuracy: 0.7628
Epoch 22/100
176/176 [==============================] - 22s 127ms/step - loss: 1.9066 - accuracy: 0.4839 -
top-5-accuracy: 0.7904 - val_loss: 1.9961 - val_accuracy: 0.4746 - val_top-5-accuracy: 0.7656
Epoch 23/100
176/176 [==============================] - 22s 127ms/step - loss: 1.8564 - accuracy: 0.4952 -
top-5-accuracy: 0.8030 - val_loss: 1.9769 - val_accuracy: 0.4828 - val_top-5-accuracy: 0.7742
Epoch 24/100
176/176 [==============================] - 22s 128ms/step - loss: 1.8167 - accuracy: 0.5034 -
top-5-accuracy: 0.8099 - val_loss: 1.9730 - val_accuracy: 0.4766 - val_top-5-accuracy: 0.7728
Epoch 25/100
176/176 [==============================] - 22s 128ms/step - loss: 1.7788 - accuracy: 0.5124 -
top-5-accuracy: 0.8174 - val_loss: 1.9187 - val_accuracy: 0.4926 - val_top-5-accuracy: 0.7854
Epoch 26/100
```

```
176/176 [==============================] - 23s 128ms/step - loss: 1.7437 - accuracy: 0.5187 -
top-5-accuracy: 0.8206 - val_loss: 1.9732 - val_accuracy: 0.4792 - val_top-5-accuracy: 0.7772
Epoch 27/100
176/176 [==============================] - 23s 128ms/step - loss: 1.6929 - accuracy: 0.5300 -
top-5-accuracy: 0.8287 - val_loss: 1.9109 - val_accuracy: 0.4928 - val_top-5-accuracy: 0.7912
Epoch 28/100
176/176 [==============================] - 23s 129ms/step - loss: 1.6647 - accuracy: 0.5400 -
top-5-accuracy: 0.8362 - val_loss: 1.9031 - val_accuracy: 0.4984 - val_top-5-accuracy: 0.7824
Epoch 29/100
176/176 [==============================] - 23s 129ms/step - loss: 1.6295 - accuracy: 0.5488 -
top-5-accuracy: 0.8402 - val_loss: 1.8744 - val_accuracy: 0.4982 - val_top-5-accuracy: 0.7910
Epoch 30/100
176/176 [==============================] - 22s 128ms/step - loss: 1.5860 - accuracy: 0.5548 -
top-5-accuracy: 0.8504 - val_loss: 1.8551 - val_accuracy: 0.5108 - val_top-5-accuracy: 0.7946
Epoch 31/100
176/176 [==============================] - 22s 127ms/step - loss: 1.5666 - accuracy: 0.5614 -
top-5-accuracy: 0.8548 - val_loss: 1.8720 - val_accuracy: 0.5076 - val_top-5-accuracy: 0.7960
Epoch 32/100
176/176 [==============================] - 22s 127ms/step - loss: 1.5272 - accuracy: 0.5712 -
top-5-accuracy: 0.8596 - val_loss: 1.8840 - val_accuracy: 0.5106 - val_top-5-accuracy: 0.7966
Epoch 33/100
176/176 [==============================] - 22s 128ms/step - loss: 1.4995 - accuracy: 0.5779 -
top-5-accuracy: 0.8651 - val_loss: 1.8660 - val_accuracy: 0.5116 - val_top-5-accuracy: 0.7904
Epoch 34/100
176/176 [==============================] - 22s 128ms/step - loss: 1.4686 - accuracy: 0.5849 -
top-5-accuracy: 0.8685 - val_loss: 1.8544 - val_accuracy: 0.5126 - val_top-5-accuracy: 0.7954
Epoch 35/100
176/176 [==============================] - 22s 127ms/step - loss: 1.4276 - accuracy: 0.5992 -
top-5-accuracy: 0.8743 - val_loss: 1.8497 - val_accuracy: 0.5164 - val_top-5-accuracy: 0.7990
Epoch 36/100
176/176 [==============================] - 22s 127ms/step - loss: 1.4102 - accuracy: 0.5970 -
top-5-accuracy: 0.8768 - val_loss: 1.8496 - val_accuracy: 0.5198 - val_top-5-accuracy: 0.7948
Epoch 37/100
176/176 [==============================] - 22s 126ms/step - loss: 1.3800 - accuracy: 0.6112 -
top-5-accuracy: 0.8814 - val_loss: 1.8033 - val_accuracy: 0.5284 - val_top-5-accuracy: 0.8068
Epoch 38/100
176/176 [==============================] - 22s 126ms/step - loss: 1.3500 - accuracy: 0.6103 -
top-5-accuracy: 0.8862 - val_loss: 1.8092 - val_accuracy: 0.5214 - val_top-5-accuracy: 0.8128
Epoch 39/100
176/176 [==============================] - 22s 127ms/step - loss: 1.3575 - accuracy: 0.6127 -
top-5-accuracy: 0.8857 - val_loss: 1.8175 - val_accuracy: 0.5198 - val_top-5-accuracy: 0.8086
Epoch 40/100
176/176 [==============================] - 22s 126ms/step - loss: 1.3030 - accuracy: 0.6283 -
top-5-accuracy: 0.8927 - val_loss: 1.8361 - val_accuracy: 0.5170 - val_top-5-accuracy: 0.8056
Epoch 41/100
176/176 [==============================] - 22s 125ms/step - loss: 1.3160 - accuracy: 0.6247 -
top-5-accuracy: 0.8923 - val_loss: 1.8074 - val_accuracy: 0.5260 - val_top-5-accuracy: 0.8082
Epoch 42/100
176/176 [==============================] - 22s 126ms/step - loss: 1.2679 - accuracy: 0.6329 -
top-5-accuracy: 0.9002 - val_loss: 1.8430 - val_accuracy: 0.5244 - val_top-5-accuracy: 0.8100
Epoch 43/100
176/176 [==============================] - 22s 126ms/step - loss: 1.2514 - accuracy: 0.6375 -
top-5-accuracy: 0.9034 - val_loss: 1.8318 - val_accuracy: 0.5196 - val_top-5-accuracy: 0.8034
Epoch 44/100
176/176 [==============================] - 22s 126ms/step - loss: 1.2311 - accuracy: 0.6431 -
top-5-accuracy: 0.9067 - val_loss: 1.8283 - val_accuracy: 0.5218 - val_top-5-accuracy: 0.8050
Epoch 45/100
176/176 [==============================] - 22s 125ms/step - loss: 1.2073 - accuracy: 0.6484 -
top-5-accuracy: 0.9098 - val_loss: 1.8384 - val_accuracy: 0.5302 - val_top-5-accuracy: 0.8056
Epoch 46/100
176/176 [==============================] - 22s 125ms/step - loss: 1.1775 - accuracy: 0.6558 -
top-5-accuracy: 0.9117 - val_loss: 1.8409 - val_accuracy: 0.5294 - val_top-5-accuracy: 0.8078
Epoch 47/100
176/176 [==============================] - 22s 126ms/step - loss: 1.1891 - accuracy: 0.6563 -
top-5-accuracy: 0.9103 - val_loss: 1.8167 - val_accuracy: 0.5346 - val_top-5-accuracy: 0.8142
Epoch 48/100
176/176 [==============================] - 22s 127ms/step - loss: 1.1586 - accuracy: 0.6621 -
top-5-accuracy: 0.9161 - val_loss: 1.8285 - val_accuracy: 0.5314 - val_top-5-accuracy: 0.8086
Epoch 49/100
176/176 [==============================] - 22s 126ms/step - loss: 1.1586 - accuracy: 0.6634 -
top-5-accuracy: 0.9154 - val_loss: 1.8189 - val_accuracy: 0.5366 - val_top-5-accuracy: 0.8134
Epoch 50/100
176/176 [==============================] - 22s 126ms/step - loss: 1.1306 - accuracy: 0.6682 -
top-5-accuracy: 0.9199 - val_loss: 1.8442 - val_accuracy: 0.5254 - val_top-5-accuracy: 0.8096
Epoch 51/100
176/176 [==============================] - 22s 126ms/step - loss: 1.1175 - accuracy: 0.6708 -
top-5-accuracy: 0.9227 - val_loss: 1.8513 - val_accuracy: 0.5230 - val_top-5-accuracy: 0.8104
```

```
Epoch 52/100
176/176 [==============================] - 22s 126ms/step - loss: 1.1104 - accuracy: 0.6743 - top-5-accuracy: 0.9226 - val_loss: 1.8041 - val_accuracy: 0.5332 - val_top-5-accuracy: 0.8142
Epoch 53/100
176/176 [==============================] - 22s 127ms/step - loss: 1.0914 - accuracy: 0.6809 - top-5-accuracy: 0.9236 - val_loss: 1.8213 - val_accuracy: 0.5342 - val_top-5-accuracy: 0.8094
Epoch 54/100
176/176 [==============================] - 22s 126ms/step - loss: 1.0681 - accuracy: 0.6856 - top-5-accuracy: 0.9270 - val_loss: 1.8429 - val_accuracy: 0.5328 - val_top-5-accuracy: 0.8086
Epoch 55/100
176/176 [==============================] - 22s 126ms/step - loss: 1.0625 - accuracy: 0.6862 - top-5-accuracy: 0.9301 - val_loss: 1.8316 - val_accuracy: 0.5364 - val_top-5-accuracy: 0.8090
Epoch 56/100
176/176 [==============================] - 22s 127ms/step - loss: 1.0474 - accuracy: 0.6920 - top-5-accuracy: 0.9308 - val_loss: 1.8310 - val_accuracy: 0.5440 - val_top-5-accuracy: 0.8132
Epoch 57/100
176/176 [==============================] - 22s 127ms/step - loss: 1.0381 - accuracy: 0.6974 - top-5-accuracy: 0.9297 - val_loss: 1.8447 - val_accuracy: 0.5368 - val_top-5-accuracy: 0.8126
Epoch 58/100
176/176 [==============================] - 22s 126ms/step - loss: 1.0230 - accuracy: 0.7011 - top-5-accuracy: 0.9341 - val_loss: 1.8241 - val_accuracy: 0.5418 - val_top-5-accuracy: 0.8094
Epoch 59/100
176/176 [==============================] - 22s 127ms/step - loss: 1.0113 - accuracy: 0.7023 - top-5-accuracy: 0.9361 - val_loss: 1.8216 - val_accuracy: 0.5380 - val_top-5-accuracy: 0.8134
Epoch 60/100
176/176 [==============================] - 22s 126ms/step - loss: 0.9953 - accuracy: 0.7031 - top-5-accuracy: 0.9386 - val_loss: 1.8356 - val_accuracy: 0.5422 - val_top-5-accuracy: 0.8122
Epoch 61/100
176/176 [==============================] - 22s 126ms/step - loss: 0.9928 - accuracy: 0.7084 - top-5-accuracy: 0.9375 - val_loss: 1.8514 - val_accuracy: 0.5342 - val_top-5-accuracy: 0.8182
Epoch 62/100
176/176 [==============================] - 22s 126ms/step - loss: 0.9740 - accuracy: 0.7121 - top-5-accuracy: 0.9387 - val_loss: 1.8674 - val_accuracy: 0.5366 - val_top-5-accuracy: 0.8092
Epoch 63/100
176/176 [==============================] - 22s 126ms/step - loss: 0.9742 - accuracy: 0.7112 - top-5-accuracy: 0.9413 - val_loss: 1.8274 - val_accuracy: 0.5414 - val_top-5-accuracy: 0.8144
Epoch 64/100
176/176 [==============================] - 22s 126ms/step - loss: 0.9633 - accuracy: 0.7147 - top-5-accuracy: 0.9393 - val_loss: 1.8250 - val_accuracy: 0.5434 - val_top-5-accuracy: 0.8180
Epoch 65/100
176/176 [==============================] - 22s 126ms/step - loss: 0.9407 - accuracy: 0.7221 - top-5-accuracy: 0.9444 - val_loss: 1.8456 - val_accuracy: 0.5424 - val_top-5-accuracy: 0.8120
Epoch 66/100
176/176 [==============================] - 22s 126ms/step - loss: 0.9410 - accuracy: 0.7194 - top-5-accuracy: 0.9447 - val_loss: 1.8559 - val_accuracy: 0.5460 - val_top-5-accuracy: 0.8144
Epoch 67/100
176/176 [==============================] - 22s 126ms/step - loss: 0.9359 - accuracy: 0.7252 - top-5-accuracy: 0.9421 - val_loss: 1.8352 - val_accuracy: 0.5458 - val_top-5-accuracy: 0.8110
Epoch 68/100
176/176 [==============================] - 22s 126ms/step - loss: 0.9232 - accuracy: 0.7254 - top-5-accuracy: 0.9460 - val_loss: 1.8479 - val_accuracy: 0.5444 - val_top-5-accuracy: 0.8132
Epoch 69/100
176/176 [==============================] - 22s 126ms/step - loss: 0.9138 - accuracy: 0.7283 - top-5-accuracy: 0.9456 - val_loss: 1.8697 - val_accuracy: 0.5312 - val_top-5-accuracy: 0.8052
Epoch 70/100
176/176 [==============================] - 22s 126ms/step - loss: 0.9095 - accuracy: 0.7295 - top-5-accuracy: 0.9478 - val_loss: 1.8550 - val_accuracy: 0.5376 - val_top-5-accuracy: 0.8170
Epoch 71/100
176/176 [==============================] - 22s 126ms/step - loss: 0.8945 - accuracy: 0.7332 - top-5-accuracy: 0.9504 - val_loss: 1.8286 - val_accuracy: 0.5436 - val_top-5-accuracy: 0.8198
Epoch 72/100
176/176 [==============================] - 22s 125ms/step - loss: 0.8936 - accuracy: 0.7344 - top-5-accuracy: 0.9479 - val_loss: 1.8727 - val_accuracy: 0.5438 - val_top-5-accuracy: 0.8182
Epoch 73/100
176/176 [==============================] - 22s 126ms/step - loss: 0.8775 - accuracy: 0.7355 - top-5-accuracy: 0.9510 - val_loss: 1.8522 - val_accuracy: 0.5404 - val_top-5-accuracy: 0.8170
Epoch 74/100
176/176 [==============================] - 22s 126ms/step - loss: 0.8660 - accuracy: 0.7390 - top-5-accuracy: 0.9513 - val_loss: 1.8432 - val_accuracy: 0.5448 - val_top-5-accuracy: 0.8156
Epoch 75/100
176/176 [==============================] - 22s 126ms/step - loss: 0.8583 - accuracy: 0.7441 - top-5-accuracy: 0.9532 - val_loss: 1.8419 - val_accuracy: 0.5462 - val_top-5-accuracy: 0.8226
Epoch 76/100
176/176 [==============================] - 22s 126ms/step - loss: 0.8549 - accuracy: 0.7443 - top-5-accuracy: 0.9529 - val_loss: 1.8757 - val_accuracy: 0.5454 - val_top-5-accuracy: 0.8086
Epoch 77/100
176/176 [==============================] - 22s 125ms/step - loss: 0.8578 - accuracy: 0.7384 -
```

```
top-5-accuracy: 0.9531 - val_loss: 1.9051 - val_accuracy: 0.5462 - val_top-5-accuracy: 0.8136
Epoch 78/100
176/176 [==============================] - 22s 125ms/step - loss: 0.8530 - accuracy: 0.7442 -
top-5-accuracy: 0.9526 - val_loss: 1.8496 - val_accuracy: 0.5384 - val_top-5-accuracy: 0.8124
Epoch 79/100
176/176 [==============================] - 22s 125ms/step - loss: 0.8403 - accuracy: 0.7485 -
top-5-accuracy: 0.9542 - val_loss: 1.8701 - val_accuracy: 0.5550 - val_top-5-accuracy: 0.8228
Epoch 80/100
176/176 [==============================] - 22s 126ms/step - loss: 0.8410 - accuracy: 0.7491 -
top-5-accuracy: 0.9538 - val_loss: 1.8737 - val_accuracy: 0.5502 - val_top-5-accuracy: 0.8150
Epoch 81/100
176/176 [==============================] - 22s 126ms/step - loss: 0.8275 - accuracy: 0.7547 -
top-5-accuracy: 0.9532 - val_loss: 1.8391 - val_accuracy: 0.5534 - val_top-5-accuracy: 0.8156
Epoch 82/100
176/176 [==============================] - 22s 125ms/step - loss: 0.8221 - accuracy: 0.7528 -
top-5-accuracy: 0.9562 - val_loss: 1.8775 - val_accuracy: 0.5428 - val_top-5-accuracy: 0.8120
Epoch 83/100
176/176 [==============================] - 22s 125ms/step - loss: 0.8270 - accuracy: 0.7526 -
top-5-accuracy: 0.9550 - val_loss: 1.8464 - val_accuracy: 0.5468 - val_top-5-accuracy: 0.8148
Epoch 84/100
176/176 [==============================] - 22s 126ms/step - loss: 0.8080 - accuracy: 0.7551 -
top-5-accuracy: 0.9576 - val_loss: 1.8789 - val_accuracy: 0.5486 - val_top-5-accuracy: 0.8204
Epoch 85/100
176/176 [==============================] - 22s 125ms/step - loss: 0.8058 - accuracy: 0.7593 -
top-5-accuracy: 0.9573 - val_loss: 1.8691 - val_accuracy: 0.5446 - val_top-5-accuracy: 0.8156
Epoch 86/100
176/176 [==============================] - 22s 126ms/step - loss: 0.8092 - accuracy: 0.7564 -
top-5-accuracy: 0.9560 - val_loss: 1.8588 - val_accuracy: 0.5524 - val_top-5-accuracy: 0.8172
Epoch 87/100
176/176 [==============================] - 22s 125ms/step - loss: 0.7897 - accuracy: 0.7613 -
top-5-accuracy: 0.9604 - val_loss: 1.8649 - val_accuracy: 0.5490 - val_top-5-accuracy: 0.8166
Epoch 88/100
176/176 [==============================] - 22s 126ms/step - loss: 0.7890 - accuracy: 0.7635 -
top-5-accuracy: 0.9598 - val_loss: 1.9060 - val_accuracy: 0.5446 - val_top-5-accuracy: 0.8112
Epoch 89/100
176/176 [==============================] - 22s 126ms/step - loss: 0.7682 - accuracy: 0.7687 -
top-5-accuracy: 0.9620 - val_loss: 1.8645 - val_accuracy: 0.5474 - val_top-5-accuracy: 0.8150
Epoch 90/100
176/176 [==============================] - 22s 125ms/step - loss: 0.7958 - accuracy: 0.7617 -
top-5-accuracy: 0.9600 - val_loss: 1.8549 - val_accuracy: 0.5496 - val_top-5-accuracy: 0.8140
Epoch 91/100
176/176 [==============================] - 22s 125ms/step - loss: 0.7978 - accuracy: 0.7603 -
top-5-accuracy: 0.9590 - val_loss: 1.9169 - val_accuracy: 0.5440 - val_top-5-accuracy: 0.8140
Epoch 92/100
176/176 [==============================] - 22s 125ms/step - loss: 0.7898 - accuracy: 0.7630 -
top-5-accuracy: 0.9594 - val_loss: 1.9015 - val_accuracy: 0.5540 - val_top-5-accuracy: 0.8174
Epoch 93/100
176/176 [==============================] - 22s 125ms/step - loss: 0.7550 - accuracy: 0.7722 -
top-5-accuracy: 0.9622 - val_loss: 1.9219 - val_accuracy: 0.5410 - val_top-5-accuracy: 0.8098
Epoch 94/100
176/176 [==============================] - 22s 125ms/step - loss: 0.7692 - accuracy: 0.7689 -
top-5-accuracy: 0.9599 - val_loss: 1.8928 - val_accuracy: 0.5506 - val_top-5-accuracy: 0.8184
Epoch 95/100
176/176 [==============================] - 22s 126ms/step - loss: 0.7783 - accuracy: 0.7661 -
top-5-accuracy: 0.9597 - val_loss: 1.8646 - val_accuracy: 0.5490 - val_top-5-accuracy: 0.8166
Epoch 96/100
176/176 [==============================] - 22s 125ms/step - loss: 0.7547 - accuracy: 0.7711 -
top-5-accuracy: 0.9638 - val_loss: 1.9347 - val_accuracy: 0.5484 - val_top-5-accuracy: 0.8150
Epoch 97/100
176/176 [==============================] - 22s 125ms/step - loss: 0.7603 - accuracy: 0.7692 -
top-5-accuracy: 0.9616 - val_loss: 1.8966 - val_accuracy: 0.5522 - val_top-5-accuracy: 0.8144
Epoch 98/100
176/176 [==============================] - 22s 125ms/step - loss: 0.7595 - accuracy: 0.7730 -
top-5-accuracy: 0.9610 - val_loss: 1.8728 - val_accuracy: 0.5470 - val_top-5-accuracy: 0.8170
Epoch 99/100
176/176 [==============================] - 22s 125ms/step - loss: 0.7542 - accuracy: 0.7736 -
top-5-accuracy: 0.9622 - val_loss: 1.9132 - val_accuracy: 0.5504 - val_top-5-accuracy: 0.8156
Epoch 100/100
176/176 [==============================] - 22s 125ms/step - loss: 0.7410 - accuracy: 0.7787 -
top-5-accuracy: 0.9635 - val_loss: 1.9233 - val_accuracy: 0.5428 - val_top-5-accuracy: 0.8120
313/313 [==============================] - 4s 12ms/step - loss: 1.8487 - accuracy: 0.5514 -
top-5-accuracy: 0.8186
Test accuracy: 55.14%
Test top 5 accuracy: 81.86%
```

After 100 epochs, the ViT model achieves around 55% accuracy and 82% top-5 accuracy on the test data. These are not competitive results on the CIFAR-100 dataset, as a ResNet50V2 trained from scratch on the same data can achieve 67% accuracy.

Note that the state of the art results reported in the paper are achieved by pre-training the ViT model using the JFT-300M dataset, then fine-tuning it on the target dataset. To improve the model quality without pre-training, you can try to train the model for more epochs, use a larger number of Transformer layers, resize the input images, change the patch size, or increase the projection dimensions. Besides, as mentioned in the paper, the quality of the model is affected not only by architecture choices, but also by parameters such as the learning rate schedule, optimizer, weight decay, etc. In practice, it's recommended to fine-tune a ViT model that was pre-trained using a large, high-resolution dataset.