Image classification with Vision Transformer

Author: Ramandeep Kaur **Date created:** 2024/07/20 **Last modified:** 2024/09/05 **Description:** Implementing the Vision Transformer (ViT) model for image classification.

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.

Setup

```
import os

os.environ["KERAS_BACKEND"] = "tensorflow" # @param ["tensorflow",
"jax", "torch"]

import keras
from keras import layers
from keras import ops

import numpy as np
import matplotlib.pyplot as plt
```

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}")
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-100-
python.tar.gz
169001437/169001437 ━━━━━━━━━━━━━━━━━━━━ 6s 0us/step
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 = 10 # For real training, use num_epochs=100. 10 is a test
value
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=keras.activations.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x
```

Implement patch creation as a layer

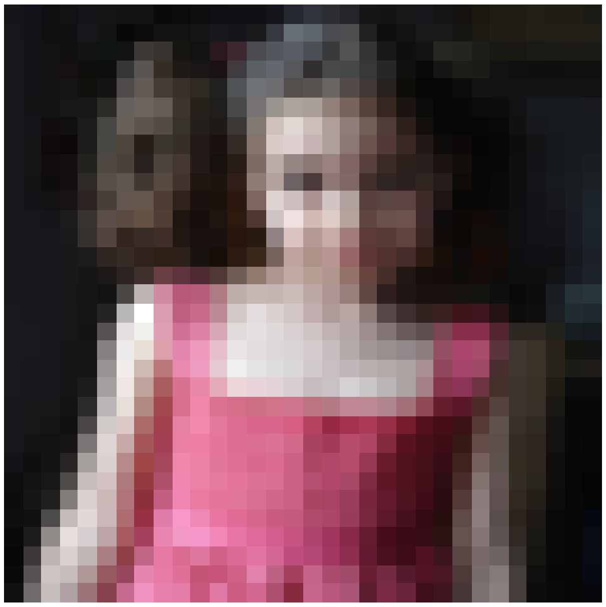
```
class Patches(layers.Layer):
    def __init__(self, patch_size):
        super().__init__()
        self.patch_size = patch_size

    def call(self, images):
        input_shape = ops.shape(images)
        batch_size = input_shape[0]
        height = input_shape[1]
        width = input_shape[2]
        channels = input_shape[3]
        num_patches_h = height // self.patch_size
        num_patches_w = width // self.patch_size
        patches = keras.ops.image.extract_patches(images,
size=self.patch_size)
        patches = ops.reshape(
            patches,
            (
                batch_size,
                num_patches_h * num_patches_w,
                self.patch_size * self.patch_size * channels,
            ),
        )
        return patches

    def get_config(self):
        config = super().get_config()
        config.update({"patch_size": self.patch_size})
        return config
```

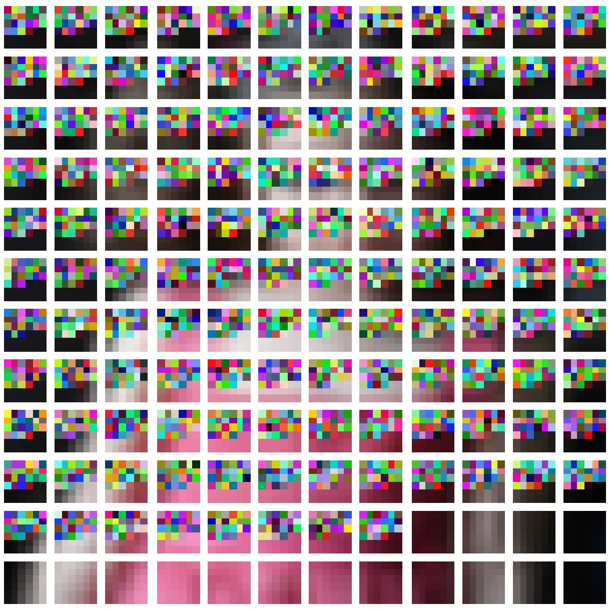
Let's display patches for a sample image

```
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 = ops.image.resize(
    ops.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 = ops.reshape(patch, (patch_size, patch_size, 3))
    plt.imshow(ops.convert_to_numpy(patch_img).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().__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 = ops.expand_dims(
            ops.arange(start=0, stop=self.num_patches, step=1), axis=0
        )
        projected_patches = self.projection(patch)
        #print("projected_patches", projected_patches)
        #print("self.position_embedding(positions)",
self.position_embedding(positions))

        encoded = projected_patches +
self.position_embedding(positions)
        return encoded

    def get_config(self):
        config = super().get_config()
        config.update({"num_patches": self.num_patches})
        return config
```

Orthonormal constraint

```
import tensorflow as tf
class Orthonormal(tf.keras.constraints.Constraint):
    """approximate Orthonormal weight constraint.
    Constrains the weights incident to each hidden unit
    to be approximately orthonormal

    # Arguments
        beta: the strength of the constraint

    # References
        https://arxiv.org/pdf/1710.04087.pdf
    """

    def __init__(self, beta=0.01):
        self.beta = beta

    def __call__(self, w):
        eye = tf.linalg.matmul(w, w, transpose_b=True)
        return (1 + self.beta) * w - self.beta * tf.linalg.matmul(eye, 
w)

    def get_config(self):
        return {'beta': self.beta}
```

Implementing the attentions

```


class DotProductAttention(layers.Layer):
    def __init__(self, normalization='standard', **kwargs):
        super().__init__(**kwargs)
        self.normalization = normalization

    def call(self, queries, keys, values, d_k, mask=None):
        scores = ops.matmul(queries, ops.transpose(keys, (0, 1, 3, 
2)))
        if self.normalization == 'linear':
            scores /= d_k
        elif self.normalization == 'cosine':
            scores /= ops.sqrt(ops.cast(d_k, "float32"))
        else:
            scores /= ops.sqrt(ops.cast(d_k, "float32"))

        if mask is not None:
            scores += -1e9 * mask
        weights = ops.softmax(scores)
        return ops.matmul(weights, values)

class CosineAttention(layers.Layer):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    def call(self, queries, keys, values, d_k, mask=None):

        # Normalize the input vectors first
        #queries_norm = ops.normalize(queries, axis=-1)
        #keys_norm = ops.normalize(keys, axis=-1)
        # Compute cosine similarity scores
        scores = ops.matmul(queries, ops.transpose(keys, (0, 1, 3, 
2)))
        # Apply mask if provided
        if mask is not None:
            scores += -1e9 * mask
        # Use scores directly as weights (no softmax)
        return ops.matmul(scores, values)

class MultiHeadAttention(layers.Layer):
    def __init__(self, h, d_k, d_v, d_model, normalization='standard',
**kwargs):
        super().__init__(**kwargs)
        self.attention =
DotProductAttention(normalization=normalization)
        self.heads = h
        self.d_k = d_k
        self.d_v = d_v
        self.d_model = d_model
        self.W_q = layers.Dense(d_k,
kernel_initializer=keras.initializers.GlorotUniform())
        self.W_k = layers.Dense(d_k,
kernel_initializer=keras.initializers.GlorotUniform())
        self.W_v = layers.Dense(d_v,
kernel_initializer=keras.initializers.GlorotUniform())
        self.W_o = layers.Dense(d_model,
kernel_initializer=keras.initializers.GlorotUniform())

    def reshape_tensor(self, x, heads, flag):
        if flag:
            lastdim = int(ops.shape(x)[2] / self.heads)
            x = ops.reshape(x, (ops.shape(x)[0], ops.shape(x)[1],
heads, lastdim))
            x = ops.transpose(x, (0, 2, 1, 3))
        else:
            x = ops.transpose(x, (0, 2, 1, 3))
            lastdim = ops.shape(x)[2] * ops.shape(x)[3]
            x = ops.reshape(x, (ops.shape(x)[0], ops.shape(x)[1],
lastdim))
        return x

    def call(self, queries, keys, values, mask=None):
        q_reshaped = self.reshape_tensor(self.W_q(queries),
self.heads, True)
        k_reshaped = self.reshape_tensor(self.W_k(keys), self.heads,
True)
        v_reshaped = self.reshape_tensor(self.W_v(values), self.heads,
True)
        o_reshaped = self.attention(queries=q_reshaped,
keys=k_reshaped, values=v_reshaped, d_k=self.d_k, mask=mask)
        output = self.reshape_tensor(o_reshaped, self.heads, False)
        return self.W_o(output)

class MultiHeadCosineAttention(layers.Layer):
    def __init__(self, h, d_k, d_v, d_model, **kwargs):
        super().__init__(**kwargs)
        self.attention = CosineAttention()
        self.heads = h
        self.d_k = d_k
        self.d_v = d_v
        self.d_model = d_model
        self.W_q = layers.Dense(d_k,
kernel_initializer=keras.initializers.GlorotUniform(),
kernel_constraint=Orthonormal())
        #self.W_k = layers.Dense(d_k,
kernel_initializer=keras.initializers.GlorotUniform())
        self.W_v = layers.Dense(d_v,
kernel_initializer=keras.initializers.GlorotUniform())
        self.W_o = layers.Dense(d_model,
kernel_initializer=keras.initializers.GlorotUniform())

    def reshape_tensor(self, x, heads, flag):
        if flag:
            lastdim = int(ops.shape(x)[2] / self.heads)
            x = ops.reshape(x, (ops.shape(x)[0], ops.shape(x)[1],
heads, lastdim))
            x = ops.transpose(x, (0, 2, 1, 3))
        else:
            x = ops.transpose(x, (0, 2, 1, 3))
            lastdim = ops.shape(x)[2] * ops.shape(x)[3]
            x = ops.reshape(x, (ops.shape(x)[0], ops.shape(x)[1],
lastdim))
        return x

    def call(self, queries, keys, values, mask=None):
        # Apply custom normalization as per teacher's feedback
        #bpt queries_norm = ops.l2_normalize(queries, axis=-1)
        #bpt keys_norm = ops.l2_normalize(keys, axis=-1)
        queries_norm = ops.normalize(queries, axis=-1)
        keys_norm = ops.normalize(keys, axis=-1)
        q_reshaped = self.reshape_tensor(self.W_q(queries_norm),
self.heads, True)
        k_reshaped = self.reshape_tensor(keys_norm, self.heads,
True)#self.W_k(keys_norm), self.heads, True)
        v_reshaped = self.reshape_tensor(self.W_v(values), self.heads,
True)
        o_reshaped = self.attention(queries=q_reshaped,
keys=k_reshaped, values=v_reshaped, d_k=self.d_k, mask=mask)
        output = self.reshape_tensor(o_reshaped, self.heads, False)
        return self.W_o(output)

# Implementing the custom activation for the "Transformers are RNNs"
paper
def custom_activation(x):
    return ops.elu(x) + 1.0

class KernelAttention(layers.Layer):
    def __init__(self, h, d_k, d_v, d_model, **kwargs):
        super().__init__(**kwargs)
        self.heads = h
        self.d_k = d_k
        self.d_v = d_v
        self.d_model = d_model
        self.W_q = layers.Dense(d_k, activation=custom_activation)
        self.W_k = layers.Dense(d_k, activation=custom_activation)
        self.W_v = layers.Dense(d_v)

    def call(self, x):
        # Computing ki, vi, qi using the kernel function activation
        qi = self.W_q(x)
        ki = self.W_k(x)
        vi = self.W_v(x)

        # Computing N and d
        N = ops.reduce_sum(ops.matmul(ki, ops.transpose(vi, (0, 1, 3,
2))), axis=1)
        d = ops.reduce_sum(ki, axis=1)

        # Computing the attention weighted vectors
        attention_output = ops.matmul(ops.matmul(qi, N),
ops.expand_dims(vi, axis=-1)) / ops.matmul(qi, d)
        return ops.squeeze(attention_output, axis=-1)
```

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 = keras.Input(shape=input_shape)
    augmented = data_augmentation(inputs)
    patches = Patches(patch_size)(augmented)
    encoded_patches = PatchEncoder(num_patches, projection_dim)
(patches)

    for _ in range(transformer_layers):
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        attention_output = MultiHeadCosineAttention(num_heads,
projection_dim, projection_dim, projection_dim)(x1, x1, x1)
        x2 = layers.Add()([attention_output, encoded_patches])
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)
        encoded_patches = layers.Add()([x3, x2])

    representation = layers.LayerNormalization(epsilon=1e-6)
(encoded_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.5)(representation)
    features = mlp(representation, hidden_units=mlp_head_units,
dropout_rate=0.5)
    logits = layers.Dense(num_classes)(features)
    model = keras.Model(inputs=inputs, outputs=logits)
    return model
```

Compile, train, and evaluate the mode

```
# Compile, train, and evaluate the model
def run_experiment(model):
    optimizer = keras.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.weights.h5"
    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=10,  # Set epochs to 1 for quick execution
        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

# Run and plot results
vit_classifier = create_vit_classifier()
vit_classifier.summary()
history = run_experiment(vit_classifier)

# Plot history
def plot_history(item):
    plt.plot(history.history[item], label=item)
    plt.plot(history.history["val_" + item], label="val_" + item)
    plt.xlabel("Epochs")
    plt.ylabel(item)
    plt.title(f"Train and Validation {item} Over Epochs", fontsize=14)
    plt.legend()
    plt.grid()
    plt.show()

plot_history("loss")
plot_history("accuracy")
plot_history("top-5-accuracy")
```

```
Model: "functional_3"

┏━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━
┏━━━━━━━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)            ┃ Output Shape           ┃        Param #
┃ Connected to            ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━
┡━━━━━━━━━━━━━━━━━━━━━━━━━┩
│ input_layer_3           │ (None, 32, 32, 3)      │              0
│ -                       │
│ (InputLayer)            │                        │
│                         │
├─────────────────────────┼────────────────────────┼───────────────
┼─────────────────────────┤
│ data_augmentation       │ (None, 72, 72, 3)      │              7
│ input_layer_3[0][0]     │
│ (Sequential)            │                        │
│                         │
├─────────────────────────┼────────────────────────┼───────────────
┼─────────────────────────┤
│ patches_3 (Patches)     │ (None, 144, 108)       │              0
│ data_augmentation[2][…  │
```

```
├────────────────────────────┼─────────────────────────┼──────────────────┤
─────────────────────────┤
 patch_encoder_2            │ (None, 144, 64)         │           16,192
 patches_3[0][0]          │
 (PatchEncoder)             │                         │
                          │
├────────────────────────────┼─────────────────────────┼──────────────────┤
─────────────────────────┤
 layer_normalization_34     │ (None, 144, 64)         │              128
 patch_encoder_2[0][0]    │
 (LayerNormalization)       │                         │
                          │
├────────────────────────────┼─────────────────────────┼──────────────────┤
─────────────────────────┤
 multi_head_cosine_attent… │ (None, 144, 64)         │           12,480
 layer_normalization_3…   │
 (MultiHeadCosineAttentio… │                         │
 layer_normalization_3…   │
                            │                         │
 layer_normalization_3…   │
├────────────────────────────┼─────────────────────────┼──────────────────┤
─────────────────────────┤
 add_32 (Add)               │ (None, 144, 64)         │                0
 multi_head_cosine_att…   │
                            │                         │
 patch_encoder_2[0][0]    │
├────────────────────────────┼─────────────────────────┼──────────────────┤
─────────────────────────┤
 layer_normalization_35     │ (None, 144, 64)         │              128
 add_32[0][0]             │
 (LayerNormalization)       │                         │
                          │
├────────────────────────────┼─────────────────────────┼──────────────────┤
─────────────────────────┤
 dense_92 (Dense)           │ (None, 144, 128)        │            8,320
 layer_normalization_3…   │
├────────────────────────────┼─────────────────────────┼──────────────────┤
─────────────────────────┤
 dropout_38 (Dropout)       │ (None, 144, 128)        │                0
 dense_92[0][0]           │
├────────────────────────────┼─────────────────────────┼──────────────────┤
─────────────────────────┤
 dense_93 (Dense)           │ (None, 144, 64)         │            8,256
 dropout_38[0][0]         │
├────────────────────────────┼─────────────────────────┼──────────────────┤
─────────────────────────┤
 dropout_39 (Dropout)       │ (None, 144, 64)         │                0
 dense_93[0][0]           │
├────────────────────────────┼─────────────────────────┼──────────────────┤
```

```
┼─────────────────────────┤
│ add_33 (Add)               │ (None, 144, 64)         │             0 
│ dropout_39[0][0],        │
│                            │                         │
│ add_32[0][0]             │
├─────────────────────────┼─────────────────────────┼───────────────
┼─────────────────────────┤
│ layer_normalization_36     │ (None, 144, 64)         │           128 
│ add_33[0][0]             │
│ (LayerNormalization)       │                         │
│                          │
├─────────────────────────┼─────────────────────────┼───────────────
┼─────────────────────────┤
│ multi_head_cosine_attent… │ (None, 144, 64)         │        12,480 
│ layer_normalization_3…   │
│ (MultiHeadCosineAttentio… │                         │
│ layer_normalization_3…   │
│                            │                         │
│ layer_normalization_3…   │
├─────────────────────────┼─────────────────────────┼───────────────
┼─────────────────────────┤
│ add_34 (Add)               │ (None, 144, 64)         │             0 
│ multi_head_cosine_att…   │
│                            │                         │
│ add_33[0][0]             │
├─────────────────────────┼─────────────────────────┼───────────────
┼─────────────────────────┤
│ layer_normalization_37     │ (None, 144, 64)         │           128 
│ add_34[0][0]             │
│ (LayerNormalization)       │                         │
│                          │
├─────────────────────────┼─────────────────────────┼───────────────
┼─────────────────────────┤
│ dense_97 (Dense)           │ (None, 144, 128)        │         8,320 
│ layer_normalization_3…   │
├─────────────────────────┼─────────────────────────┼───────────────
┼─────────────────────────┤
│ dropout_40 (Dropout)       │ (None, 144, 128)        │             0 
│ dense_97[0][0]           │
├─────────────────────────┼─────────────────────────┼───────────────
┼─────────────────────────┤
│ dense_98 (Dense)           │ (None, 144, 64)         │         8,256 
│ dropout_40[0][0]         │
├─────────────────────────┼─────────────────────────┼───────────────
┼─────────────────────────┤
│ dropout_41 (Dropout)       │ (None, 144, 64)         │             0 
│ dense_98[0][0]           │
├─────────────────────────┼─────────────────────────┼───────────────
┼─────────────────────────┤
```

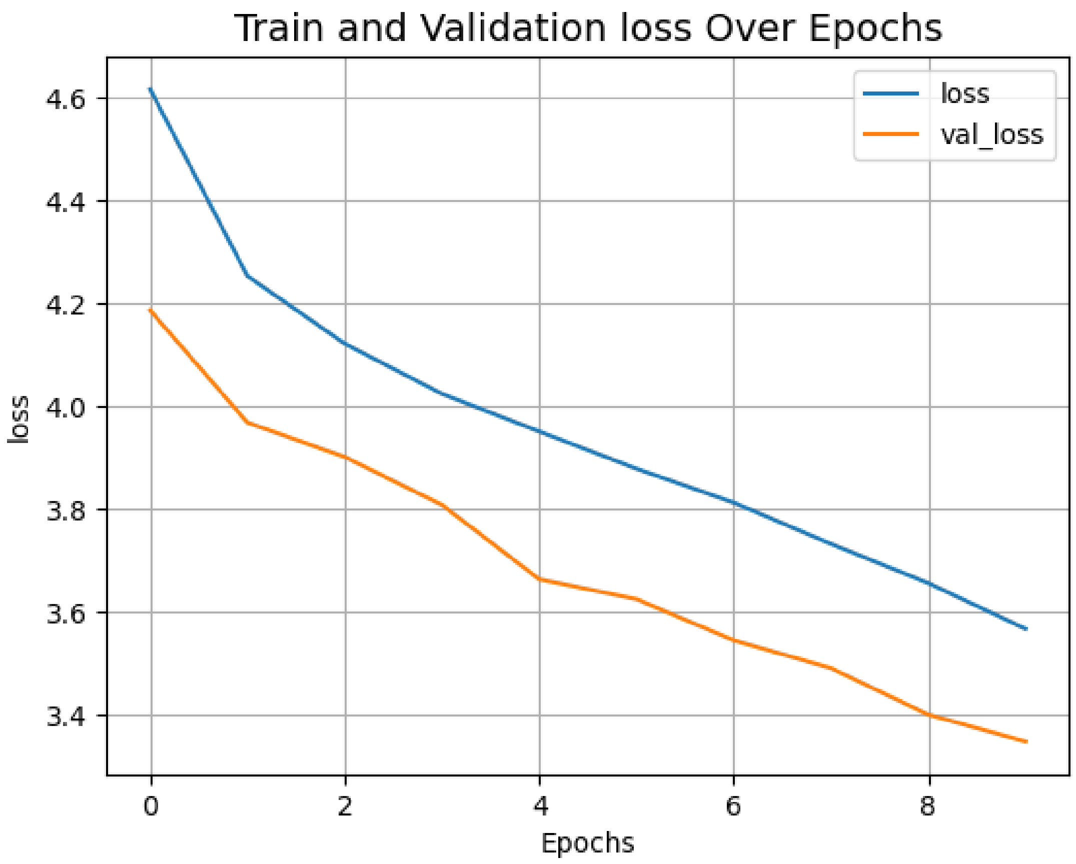
```
│ add_35 (Add)              │ (None, 144, 64)        │             0 
│ dropout_41[0][0],        │
│                           │                        │
│ add_34[0][0]             │
├───────────────────────────┼────────────────────────┼───────────────
┤────────────────────────┤
│ layer_normalization_38    │ (None, 144, 64)        │           128 
│ add_35[0][0]             │
│ (LayerNormalization)      │                        │
│                          │
├───────────────────────────┼────────────────────────┼───────────────
┤────────────────────────┤
│ multi_head_cosine_attent… │ (None, 144, 64)        │        12,480 
│ layer_normalization_3… │
│ (MultiHeadCosineAttentio… │                        │
│ layer_normalization_3… │
│                           │                        │
│ layer_normalization_3… │
├───────────────────────────┼────────────────────────┼───────────────
┤────────────────────────┤
│ add_36 (Add)              │ (None, 144, 64)        │             0 
│ multi_head_cosine_att… │
│                           │                        │
│ add_35[0][0]             │
├───────────────────────────┼────────────────────────┼───────────────
┤────────────────────────┤
│ layer_normalization_39    │ (None, 144, 64)        │           128 
│ add_36[0][0]             │
│ (LayerNormalization)      │                        │
│                          │
├───────────────────────────┼────────────────────────┼───────────────
┤────────────────────────┤
│ dense_102 (Dense)         │ (None, 144, 128)       │         8,320 
│ layer_normalization_3… │
├───────────────────────────┼────────────────────────┼───────────────
┤────────────────────────┤
│ dropout_42 (Dropout)      │ (None, 144, 128)       │             0 
│ dense_102[0][0]          │
├───────────────────────────┼────────────────────────┼───────────────
┤────────────────────────┤
│ dense_103 (Dense)         │ (None, 144, 64)        │         8,256 
│ dropout_42[0][0]         │
├───────────────────────────┼────────────────────────┼───────────────
┤────────────────────────┤
│ dropout_43 (Dropout)      │ (None, 144, 64)        │             0 
│ dense_103[0][0]          │
├───────────────────────────┼────────────────────────┼───────────────
┤────────────────────────┤
│ add_37 (Add)              │ (None, 144, 64)        │             0 
│ dropout_43[0][0],        │
                             │                         │
│ add_36[0][0]             │
├──────────────────────────┼─────────────────────────┼────────────────
┤──────────────────────────┤
│ layer_normalization_40     │ (None, 144, 64)         │            128
│ add_37[0][0]             │
│ (LayerNormalization)       │                         │
│                          │
├──────────────────────────┼─────────────────────────┼────────────────
┤──────────────────────────┤
│ multi_head_cosine_attent… │ (None, 144, 64)         │         12,480
│ layer_normalization_4… │
│ (MultiHeadCosineAttentio… │                         │
│ layer_normalization_4… │
                             │                         │
│ layer_normalization_4… │
├──────────────────────────┼─────────────────────────┼────────────────
┤──────────────────────────┤
│ add_38 (Add)               │ (None, 144, 64)         │              0
│ multi_head_cosine_att… │
                             │                         │
│ add_37[0][0]             │
├──────────────────────────┼─────────────────────────┼────────────────
┤──────────────────────────┤
│ layer_normalization_41     │ (None, 144, 64)         │            128
│ add_38[0][0]             │
│ (LayerNormalization)       │                         │
│                          │
├──────────────────────────┼─────────────────────────┼────────────────
┤──────────────────────────┤
│ dense_107 (Dense)          │ (None, 144, 128)        │          8,320
│ layer_normalization_4… │
├──────────────────────────┼─────────────────────────┼────────────────
┤──────────────────────────┤
│ dropout_44 (Dropout)       │ (None, 144, 128)        │              0
│ dense_107[0][0]          │
├──────────────────────────┼─────────────────────────┼────────────────
┤──────────────────────────┤
│ dense_108 (Dense)          │ (None, 144, 64)         │          8,256
│ dropout_44[0][0]         │
├──────────────────────────┼─────────────────────────┼────────────────
┤──────────────────────────┤
│ dropout_45 (Dropout)       │ (None, 144, 64)         │              0
│ dense_108[0][0]          │
├──────────────────────────┼─────────────────────────┼────────────────
┤──────────────────────────┤
│ add_39 (Add)               │ (None, 144, 64)         │              0
│ dropout_45[0][0],        │
│                            │                        │
add_38[0][0]              │
├────────────────────────────┼────────────────────────┼────────────────
┤
layer_normalization_42    │ (None, 144, 64)        │            128
add_39[0][0]              │
(LayerNormalization)      │                        │
                          │
├────────────────────────────┼────────────────────────┼────────────────
┤
multi_head_cosine_attent… │ (None, 144, 64)        │         12,480
layer_normalization_4…    │
(MultiHeadCosineAttentio… │                        │
layer_normalization_4…    │
                          │                        │
layer_normalization_4…    │
├────────────────────────────┼────────────────────────┼────────────────
┤
add_40 (Add)              │ (None, 144, 64)        │              0
multi_head_cosine_att…    │
                          │                        │
add_39[0][0]              │
├────────────────────────────┼────────────────────────┼────────────────
┤
layer_normalization_43    │ (None, 144, 64)        │            128
add_40[0][0]              │
(LayerNormalization)      │                        │
                          │
├────────────────────────────┼────────────────────────┼────────────────
┤
dense_112 (Dense)         │ (None, 144, 128)       │          8,320
layer_normalization_4…    │
├────────────────────────────┼────────────────────────┼────────────────
┤
dropout_46 (Dropout)      │ (None, 144, 128)       │              0
dense_112[0][0]           │
├────────────────────────────┼────────────────────────┼────────────────
┤
dense_113 (Dense)         │ (None, 144, 64)        │          8,256
dropout_46[0][0]          │
├────────────────────────────┼────────────────────────┼────────────────
┤
dropout_47 (Dropout)      │ (None, 144, 64)        │              0
dense_113[0][0]           │
├────────────────────────────┼────────────────────────┼────────────────
┤
add_41 (Add)              │ (None, 144, 64)        │              0
dropout_47[0][0],         │
│                            │                        │
│ add_40[0][0]              │
├───────────────────────────┼─────────────────────────┼────────────────┤
┤───────────────────────────┤
│ layer_normalization_44    │ (None, 144, 64)         │            128
│ add_41[0][0]              │
│ (LayerNormalization)      │                         │
│                           │
├───────────────────────────┼─────────────────────────┼────────────────┤
┤───────────────────────────┤
│ multi_head_cosine_attent… │ (None, 144, 64)         │         12,480
│ layer_normalization_4…    │
│ (MultiHeadCosineAttentio… │                         │
│ layer_normalization_4…    │
│                           │                         │
│ layer_normalization_4…    │
├───────────────────────────┼─────────────────────────┼────────────────┤
┤───────────────────────────┤
│ add_42 (Add)              │ (None, 144, 64)         │              0
│ multi_head_cosine_att…    │
│                           │                         │
│ add_41[0][0]              │
├───────────────────────────┼─────────────────────────┼────────────────┤
┤───────────────────────────┤
│ layer_normalization_45    │ (None, 144, 64)         │            128
│ add_42[0][0]              │
│ (LayerNormalization)      │                         │
│                           │
├───────────────────────────┼─────────────────────────┼────────────────┤
┤───────────────────────────┤
│ dense_117 (Dense)         │ (None, 144, 128)        │          8,320
│ layer_normalization_4…    │
├───────────────────────────┼─────────────────────────┼────────────────┤
┤───────────────────────────┤
│ dropout_48 (Dropout)      │ (None, 144, 128)        │              0
│ dense_117[0][0]           │
├───────────────────────────┼─────────────────────────┼────────────────┤
┤───────────────────────────┤
│ dense_118 (Dense)         │ (None, 144, 64)         │          8,256
│ dropout_48[0][0]          │
├───────────────────────────┼─────────────────────────┼────────────────┤
┤───────────────────────────┤
│ dropout_49 (Dropout)      │ (None, 144, 64)         │              0
│ dense_118[0][0]           │
├───────────────────────────┼─────────────────────────┼────────────────┤
┤───────────────────────────┤
│ add_43 (Add)              │ (None, 144, 64)         │              0
│ dropout_49[0][0],         │
│                           │                         │
│ add_42[0][0]              │
```

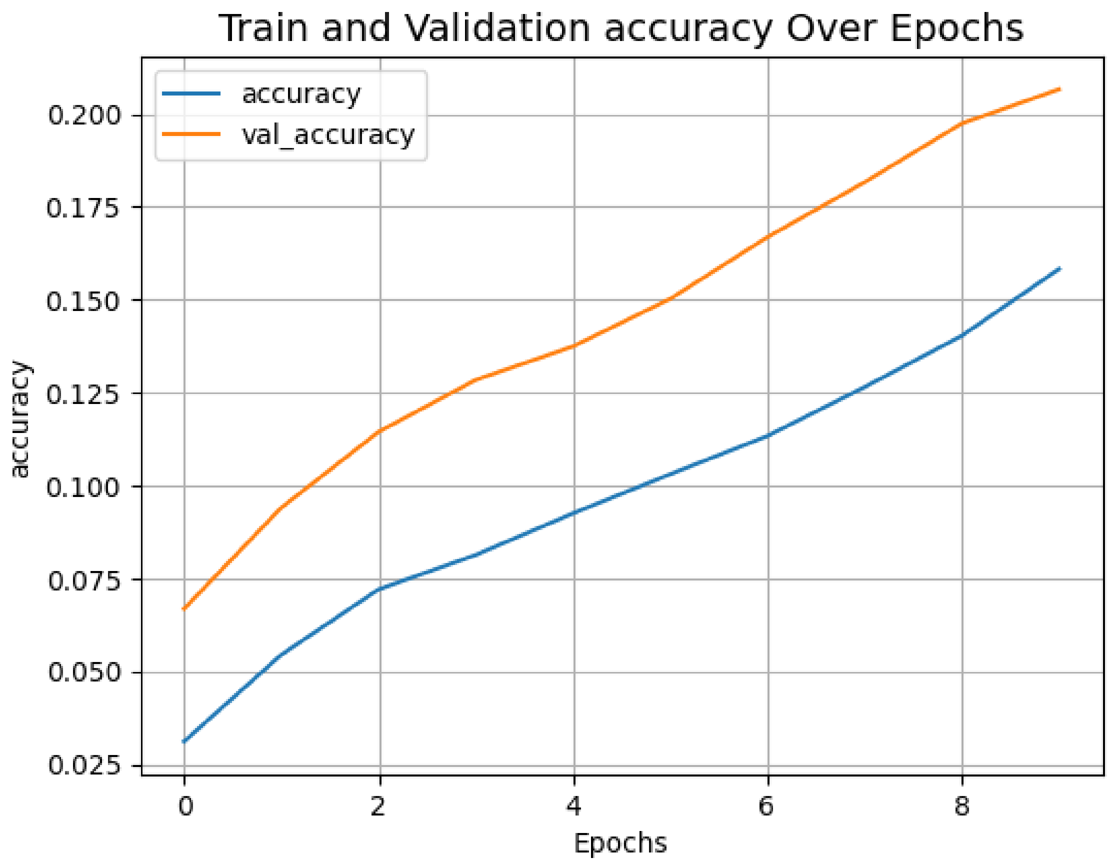
```
├────────────────────────────┼─────────────────────────┼──────────────
┤
 layer_normalization_46     │ (None, 144, 64)         │          128
 add_43[0][0]              │
 (LayerNormalization)       │                         │
                           │
├────────────────────────────┼─────────────────────────┼──────────────
┤
 multi_head_cosine_attent…  │ (None, 144, 64)         │       12,480
 layer_normalization_4…    │
 (MultiHeadCosineAttentio…  │                         │
 layer_normalization_4…    │
                            │                         │
 layer_normalization_4…    │
├────────────────────────────┼─────────────────────────┼──────────────
┤
 add_44 (Add)               │ (None, 144, 64)         │            0
 multi_head_cosine_att…    │
                            │                         │
 add_43[0][0]              │
├────────────────────────────┼─────────────────────────┼──────────────
┤
 layer_normalization_47     │ (None, 144, 64)         │          128
 add_44[0][0]              │
 (LayerNormalization)       │                         │
                           │
├────────────────────────────┼─────────────────────────┼──────────────
┤
 dense_122 (Dense)          │ (None, 144, 128)        │        8,320
 layer_normalization_4…    │
├────────────────────────────┼─────────────────────────┼──────────────
┤
 dropout_50 (Dropout)       │ (None, 144, 128)        │            0
 dense_122[0][0]           │
├────────────────────────────┼─────────────────────────┼──────────────
┤
 dense_123 (Dense)          │ (None, 144, 64)         │        8,256
 dropout_50[0][0]          │
├────────────────────────────┼─────────────────────────┼──────────────
┤
 dropout_51 (Dropout)       │ (None, 144, 64)         │            0
 dense_123[0][0]           │
├────────────────────────────┼─────────────────────────┼──────────────
┤
 add_45 (Add)               │ (None, 144, 64)         │            0
 dropout_51[0][0],         │
                            │                         │
 add_44[0][0]              │
├────────────────────────────┼─────────────────────────┼──────────────
```

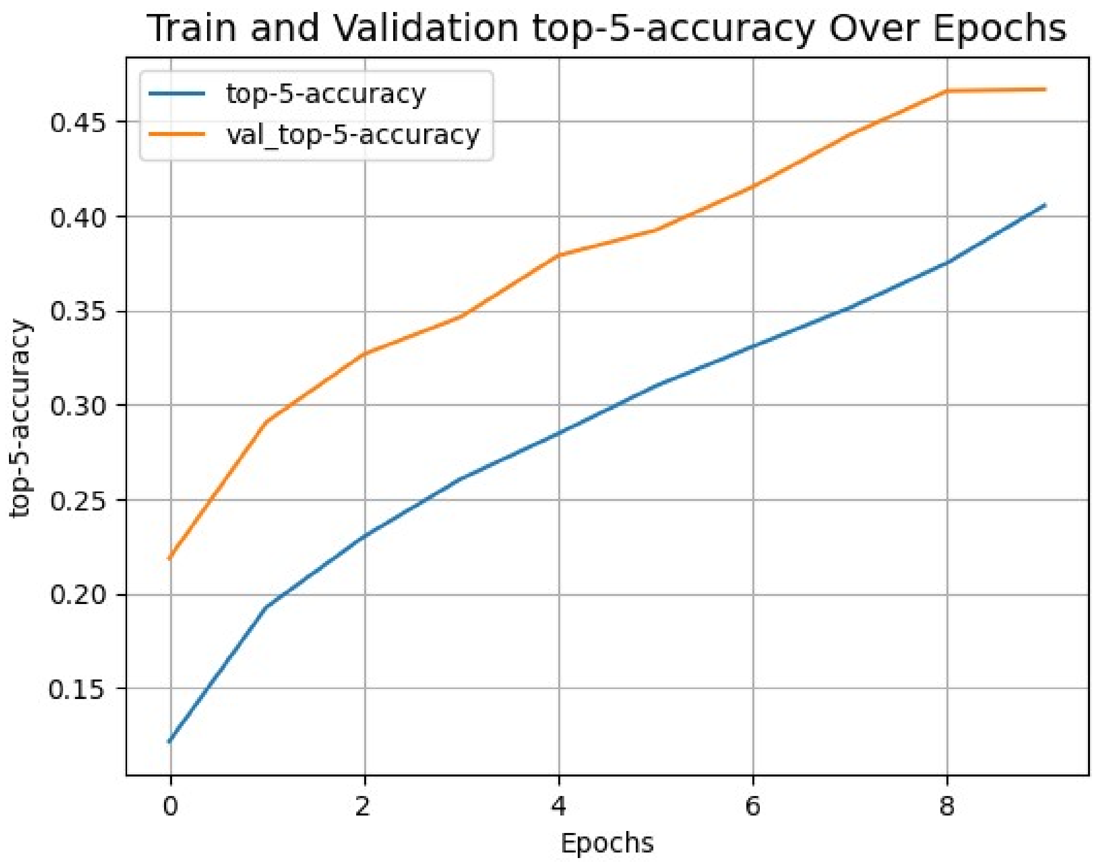
```
├─────────────────────────┤
│ layer_normalization_48  │ (None, 144, 64)        │        128
│ add_45[0][0]            │
│ (LayerNormalization)    │                        │
│                         │
├─────────────────────────┼────────────────────────┼────────────
├─────────────────────────┤
│ multi_head_cosine_attent… │ (None, 144, 64)      │     12,480
│ layer_normalization_4…  │
│ (MultiHeadCosineAttentio… │                      │
│ layer_normalization_4…  │
│                         │                        │
│ layer_normalization_4…  │
├─────────────────────────┼────────────────────────┼────────────
├─────────────────────────┤
│ add_46 (Add)            │ (None, 144, 64)        │          0
│ multi_head_cosine_att…  │
│                         │                        │
│ add_45[0][0]            │
├─────────────────────────┼────────────────────────┼────────────
├─────────────────────────┤
│ layer_normalization_49  │ (None, 144, 64)        │        128
│ add_46[0][0]            │
│ (LayerNormalization)    │                        │
│                         │
├─────────────────────────┼────────────────────────┼────────────
├─────────────────────────┤
│ dense_127 (Dense)       │ (None, 144, 128)       │      8,320
│ layer_normalization_4…  │
├─────────────────────────┼────────────────────────┼────────────
├─────────────────────────┤
│ dropout_52 (Dropout)    │ (None, 144, 128)       │          0
│ dense_127[0][0]         │
├─────────────────────────┼────────────────────────┼────────────
├─────────────────────────┤
│ dense_128 (Dense)       │ (None, 144, 64)        │      8,256
│ dropout_52[0][0]        │
├─────────────────────────┼────────────────────────┼────────────
├─────────────────────────┤
│ dropout_53 (Dropout)    │ (None, 144, 64)        │          0
│ dense_128[0][0]         │
├─────────────────────────┼────────────────────────┼────────────
├─────────────────────────┤
│ add_47 (Add)            │ (None, 144, 64)        │          0
│ dropout_53[0][0],       │
│                         │                        │
│ add_46[0][0]            │
├─────────────────────────┼────────────────────────┼────────────
├─────────────────────────┤
```

```
│ layer_normalization_50    │ (None, 144, 64)         │            128 │
│ add_47[0][0]             │
│ (LayerNormalization)      │                         │
│                          │
├───────────────────────────┼─────────────────────────┼────────────────
─┼──────────────────────────┤
│ flatten_2 (Flatten)       │ (None, 9216)            │              0 │
│ layer_normalization_5… │
├───────────────────────────┼─────────────────────────┼────────────────
─┼──────────────────────────┤
│ dropout_54 (Dropout)      │ (None, 9216)            │              0 │
│ flatten_2[0][0]          │
├───────────────────────────┼─────────────────────────┼────────────────
─┼──────────────────────────┤
│ dense_129 (Dense)         │ (None, 2048)            │     18,876,416 │
│ dropout_54[0][0]         │
├───────────────────────────┼─────────────────────────┼────────────────
─┼──────────────────────────┤
│ dropout_55 (Dropout)      │ (None, 2048)            │              0 │
│ dense_129[0][0]          │
├───────────────────────────┼─────────────────────────┼────────────────
─┼──────────────────────────┤
│ dense_130 (Dense)         │ (None, 1024)            │      2,098,176 │
│ dropout_55[0][0]         │
├───────────────────────────┼─────────────────────────┼────────────────
─┼──────────────────────────┤
│ dropout_56 (Dropout)      │ (None, 1024)            │              0 │
│ dense_130[0][0]          │
├───────────────────────────┼─────────────────────────┼────────────────
─┼──────────────────────────┤
│ dense_131 (Dense)         │ (None, 100)             │        102,500 │
│ dropout_56[0][0]         │
└───────────────────────────┴─────────────────────────┴────────────────
─┴──────────────────────────┘

 Total params: 21,327,915 (81.36 MB)

 Trainable params: 21,327,908 (81.36 MB)

 Non-trainable params: 7 (32.00 B)

Epoch 1/10
176/176 ━━━━━━━━━━━━━━━━━━━━ 73s 269ms/step - accuracy: 0.0236 - loss:
5.0342 - top-5-accuracy: 0.0931 - val_accuracy: 0.0668 - val_loss:
4.1864 - val_top-5-accuracy: 0.2186
Epoch 2/10
176/176 ━━━━━━━━━━━━━━━━━━━━ 82s 271ms/step - accuracy: 0.0527 - loss:
4.2856 - top-5-accuracy: 0.1850 - val_accuracy: 0.0940 - val_loss:
3.9684 - val_top-5-accuracy: 0.2908
Epoch 3/10
```

```
176/176 ━━━━━━━━━━━━━━━━━━━━ 112s 441ms/step - accuracy: 0.0663 -
loss: 4.1471 - top-5-accuracy: 0.2232 - val_accuracy: 0.1144 -
val_loss: 3.9018 - val_top-5-accuracy: 0.3266
Epoch 4/10
176/176 ━━━━━━━━━━━━━━━━━━━━ 49s 252ms/step - accuracy: 0.0803 - loss:
4.0291 - top-5-accuracy: 0.2600 - val_accuracy: 0.1284 - val_loss:
3.8086 - val_top-5-accuracy: 0.3464
Epoch 5/10
176/176 ━━━━━━━━━━━━━━━━━━━━ 83s 261ms/step - accuracy: 0.0924 - loss:
3.9565 - top-5-accuracy: 0.2835 - val_accuracy: 0.1374 - val_loss:
3.6643 - val_top-5-accuracy: 0.3788
Epoch 6/10
176/176 ━━━━━━━━━━━━━━━━━━━━ 45s 258ms/step - accuracy: 0.1011 - loss:
3.8905 - top-5-accuracy: 0.3057 - val_accuracy: 0.1502 - val_loss:
3.6256 - val_top-5-accuracy: 0.3922
Epoch 7/10
176/176 ━━━━━━━━━━━━━━━━━━━━ 82s 259ms/step - accuracy: 0.1132 - loss:
3.8099 - top-5-accuracy: 0.3291 - val_accuracy: 0.1668 - val_loss:
3.5456 - val_top-5-accuracy: 0.4152
Epoch 8/10
176/176 ━━━━━━━━━━━━━━━━━━━━ 82s 259ms/step - accuracy: 0.1261 - loss:
3.7330 - top-5-accuracy: 0.3535 - val_accuracy: 0.1816 - val_loss:
3.4915 - val_top-5-accuracy: 0.4428
Epoch 9/10
176/176 ━━━━━━━━━━━━━━━━━━━━ 82s 261ms/step - accuracy: 0.1378 - loss:
3.6589 - top-5-accuracy: 0.3694 - val_accuracy: 0.1974 - val_loss:
3.4013 - val_top-5-accuracy: 0.4660
Epoch 10/10
176/176 ━━━━━━━━━━━━━━━━━━━━ 43s 244ms/step - accuracy: 0.1562 - loss:
3.5774 - top-5-accuracy: 0.4016 - val_accuracy: 0.2066 - val_loss:
3.3490 - val_top-5-accuracy: 0.4668
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.2127 - loss:
3.3167 - top-5-accuracy: 0.4816
Test accuracy: 21.35%
Test top 5 accuracy: 48.1%
```


Train and Validation loss Over Epochs
loss
val_loss
4.6
4.4
4.2
4.0
3.8
3.6
3.4
loss
0
2
4
6
8
Epochs


Train and Validation accuracy Over Epochs
accuracy
val_accuracy
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0
2
4
6
8
Epochs



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.