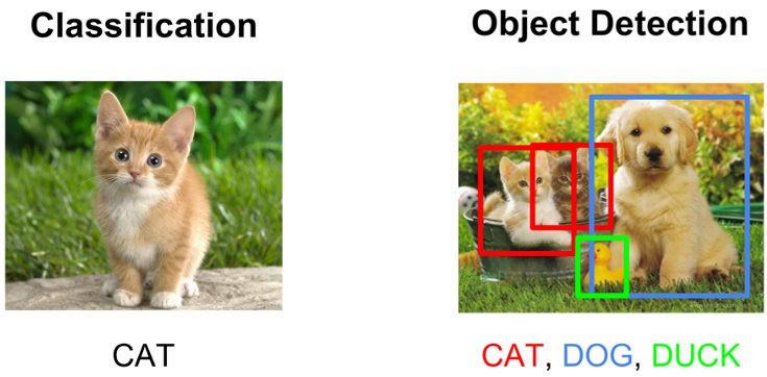
Uygulama 14a. Transformer Mimarisi İle Görüntü Sınıflandırma

1. **Amaç:**
   Bu uygulamada Vision Transformer mimarisi ile görüntü sınıflandırma işlemi yapılacaktır.

2. **Teorik Bilgi:**
   Görüntü sınıflandırma, bir bilgisayarın dijital bir görüntüyü belirli bir kategoriye veya sınıfa atama sürecidir. Bu tür bir görev genellikle makine öğrenimi ve derin öğrenme teknikleri kullanılarak gerçekleştirilir. Görüntü sınıflandırma, bir algoritmanın öğrenme süreci boyunca bir dizi etiketlenmiş görüntü verisi üzerinde eğitilmesini içerir. Bu veri seti, öğrenme algoritmasının farklı sınıfları ayırt etmesi gereken örnekleri içerir. Öğrenme süreci tamamlandıktan sonra, algoritma yeni ve etiketlenmemiş görüntülerle karşılaştığında, bu görüntüyü doğru bir şekilde sınıflandırabilir. Örnek olarak, bir görüntü sınıflandırma modeli, bir kedi veya köpek resmi gibi giriş görüntülerini analiz ederek, görüntünün bir kedi mi yoksa bir köpek mi olduğunu belirleyebilir.



Resim 1. Görüntü Sınıflandırma Örnekleri

- **K-Nearest Neighbors (K-NN):** Bu geleneksel bir makine öğrenimi algoritmasıdır. Bir görüntünün sınıfını belirlemek için, komşu noktalardan alınan etiketler kullanılır. Ancak, genellikle büyük veri setlerinde ve yüksek boyutlu özellik uzaylarında etkisiz olabilir.
- **Destek Vektör Makineleri (SVM):** SVM, veri noktalarını iki sınıfa ayıran bir hiper düzlem bulmaya çalışır. Görüntü sınıflandırmada kullanıldığında, özellik vektörleri üzerinde çalışır ve sınıflandırma yapar. Ancak, büyük veri setlerinde eğitim süreleri uzun olabilir.
- **Karar Ağaçları ve Ormanları:** Bu teknikler, veri kümesini sınıflandırmak için ağaç yapısı kullanır. Her düğüm, belirli bir özelliği değerlendirir ve bir sonraki düğüme yönlendirir. Karar ağaçları genellikle daha küçük veri setlerinde etkilidir.
- **Yapay Sinir Ağları (ANN):** Derin öğrenme yöntemlerinden biri olan yapay sinir ağları, çok katmanlı yapılarıyla karmaşık ilişkileri öğrenmeye uygun bir şekilde tasarlanmıştır. Evrişimli Sinir Ağları (CNN) ise özellikle görüntü sınıflandırmada başarılıdır. Evrişimli katmanlar, özelliklerin belirli alanlarda vurgulanmasına izin verir.
- **Evrişimli Sinir Ağları (CNN):** CNN'ler, özellikle görüntü işleme görevlerinde başarılı olan bir tür yapay sinir ağıdır. Görüntü üzerinde konvolüsyon işlemleri gerçekleştirerek özellikleri çıkarmak için tasarlanmıştır. Bu özellikler daha sonra tam

bağlantılı katmanlarda birleştirilir ve sınıflandırma yapılır.

- **Transfer Öğrenme:** Önceden eğitilmiş bir modelin ağırlıkları, yeni bir sınıflandırma görevi için başlangıç noktası olarak kullanılabilir. Bu, daha küçük veri setlerinde etkili olabilir.
- **Derin Öğrenme ve Autoencoders:** Autoencoders, veri setindeki özellikleri otomatik olarak öğrenen ve temsil eden sinir ağı modelleridir. Bu özellikler daha sonra sınıflandırma görevlerinde kullanılabilir.

3. Uygulama

1. Model oluşturmak için gerekli kütüphanelerin import edilmesi işlemi yapılmıştır.

```
import os
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"

import tensorflow as tf
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model
```

2. Ardından Transformer mimarisinin oluşturulması için gerekli olan model yapıları ve katmanları belirtilmiştir.

```
class ClassToken(Layer):
    def __init__(self):
        super().__init__()

    def build(self, input_shape):
        w_init = tf.random_normal_initializer()
        self.w = tf.Variable(
            initial_value = w_init(shape=(1, 1, input_shape[-1]), dtype=tf.float32),
            trainable = True
        )

    def call(self, inputs):
        batch_size = tf.shape(inputs)[0]
        hidden_dim = self.w.shape[-1]

        cls = tf.broadcast_to(self.w, [batch_size, 1, hidden_dim])
        cls = tf.cast(cls, dtype=inputs.dtype)
        return cls

def mlp(x, cf):
    x = Dense(cf["mlp_dim"], activation="gelu")(x)
    x = Dropout(cf["dropout_rate"])(x)
    x = Dense(cf["hidden_dim"])(x)
    x = Dropout(cf["dropout_rate"])(x)
    return x

def transformer_encoder(x, cf):
    skip_1 = x
    x = LayerNormalization()(x)
    x = MultiHeadAttention(
        num_heads=cf["num_heads"], key_dim=cf["hidden_dim"]
    )(x, x)
    x = Add()([x, skip_1])

    skip_2 = x
    x = LayerNormalization()(x)
    x = mlp(x, cf)
    x = Add()([x, skip_2])

    return x
```

3. Bir sonraki adımda Vision Transformer modeli belirtilmiştir.

```
def ViT(cf):
    input_shape = (cf["num_patches"], cf["patch_size"]*cf["patch_size"]*cf["num_channels"])
    inputs = Input(input_shape)
    """ Patch + Position Embeddings """
    patch_embed = Dense(cf["hidden_dim"])(inputs)

    positions = tf.range(start=0, limit=cf["num_patches"], delta=1)
    pos_embed = Embedding(input_dim=cf["num_patches"], output_dim=cf["hidden_dim"])(positions)
    embed = patch_embed + pos_embed

    token = ClassToken()(embed)
    x = Concatenate(axis=1)([token, embed])

    for _ in range(cf["num_layers"]):
        x = transformer_encoder(x, cf)

    x = LayerNormalization()(x)
    x = x[:, 0, :]
    x = Dense(cf["num_classes"], activation="softmax")(x)

    model = Model(inputs, x)
    return model
```

4. Model için gerekli olan sabit değişkenlerin değerleri atanmıştır. “model.summary()” satırı ile oluşturduğunuz modelin nasıl bir yapıya sahip olduğunu görebilirsiniz ve ayrıca istediğiniz gibi modeli güncelleyebilirsiniz.

```
if __name__ == "__main__":
    config = {}
    config["num_layers"] = 12
    config["hidden_dim"] = 768
    config["mlp_dim"] = 3072
    config["num_heads"] = 12
    config["dropout_rate"] = 0.1
    config["num_patches"] = 256
    config["patch_size"] = 32
    config["num_channels"] = 3
    config["num_classes"] = 5

    model = ViT(config)
    model.summary()
```

5. Eğitim için gerekli kütüphanelerin import edilmesi işlemi yapılmıştır.

```
import os
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"

import numpy as np
import cv2
from glob import glob
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from patchify import patchify
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint, CSVLogger, ReduceLROnPlateau, EarlyStopping
from vit import ViT
```

6. Hiperparametrelerin belirtildiği ve train, test, validation şeklinde veri setinin parçalandığı fonskiyonlar yazılmıştır.

```
hp = {}
hp["image_size"] = 200
hp["num_channels"] = 3
hp["patch_size"] = 25
hp["num_patches"] = (hp["image_size"]**2) // (hp["patch_size"]**2)
hp["flat_patches_shape"] = (hp["num_patches"], hp["patch_size"]*hp["patch_size"]*hp["num_channels"])

hp["batch_size"] = 32
hp["lr"] = 1e-4
hp["num_epochs"] = 500
hp["num_classes"] = 5
hp["class_names"] = ["daisy", "dandelion", "rose", "sunflower", "tulip"]

hp["num_layers"] = 12
hp["hidden_dim"] = 768
hp["mlp_dim"] = 3072
hp["num_heads"] = 12
hp["dropout_rate"] = 0.1
def create_dir(path):
    if not os.path.exists(path):
        os.makedirs(path)
def load_data(path, split=0.1):
    images = shuffle(glob(os.path.join(path, "*", "*.jpg")))
    split_size = int(len(images) * split)
    train_x, valid_x = train_test_split(images, test_size=split_size, random_state=42)
    train_x, test_x = train_test_split(train_x, test_size=split_size, random_state=42)
    return train_x, valid_x, test_x
```

7. Classification label işlemini gerçekleştiren sınıflandırma yapmak istediğimiz sınıfları belirleyen fonksiyonlar yazılmıştır.

```
def process_image_label(path):
    path = path.decode()
    image = cv2.imread(path, cv2.IMREAD_COLOR)
    image = cv2.resize(image, (hp["image_size"], hp["image_size"]))
    image = image/255.0
    patch_shape = (hp["patch_size"], hp["patch_size"], hp["num_channels"])
    patches = patchify(image, patch_shape, hp["patch_size"])
    patches = np.reshape(patches, hp["flat_patches_shape"])
    patches = patches.astype(np.float32)
    class_name = path.split("/")[-2]
    class_idx = hp["class_names"].index(class_name)
    class_idx = np.array(class_idx, dtype=np.int32)
    return patches, class_idx
def parse(path):
    patches, labels = tf.numpy_function(process_image_label, [path], [tf.float32, tf.int32])
    labels = tf.one_hot(labels, hp["num_classes"])
    patches.set_shape(hp["flat_patches_shape"])
    labels.set_shape(hp["num_classes"])
    return patches, labels
def tf_dataset(images, batch=32):
    ds = tf.data.Dataset.from_tensor_slices((images))
    ds = ds.map(parse).batch(batch).prefetch(8)
    return ds
```

8. Ardından modelin derlenip, eğitimin başlaması için gerekli kodlar eklenmiştir.

```
if __name__ == "__main__":
    np.random.seed(42)
    tf.random.set_seed(42)
    create_dir("files")
    dataset_path = "veri yolu"
    model_path = os.path.join("files", "model.h5")
    csv_path = os.path.join("files", "log.csv")
    train_x, valid_x, test_x = load_data(dataset_path)
    print(f"Train: {len(train_x)} - Valid: {len(valid_x)} - Test: {len(test_x)}")
    train_ds = tf_dataset(train_x, batch=hp["batch_size"])
    valid_ds = tf_dataset(valid_x, batch=hp["batch_size"])
    model = ViT(hp)
    model.compile(
        loss="categorical_crossentropy",
        optimizer=tf.keras.optimizers.Adam(hp["lr"], clipvalue=1.0),
        metrics=["acc"]
    )
    callbacks = [
        ModelCheckpoint(model_path, monitor='val_loss', verbose=1, save_best_only=True),
        ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=10, min_lr=1e-10, verbose=1),
        CSVLogger(csv_path),
        EarlyStopping(monitor='val_loss', patience=50, restore_best_weights=False),
    ]
    model.fit(
        train_ds,
        epochs=hp["num_epochs"],
        validation_data=valid_ds,
        callbacks=callbacks
    )
```

4. Değerlendirme Soruları:

1) Verilen kodu inceleyerek kendi modelinizi oluşturunuz.
2) Verilen epoch değerlerini değiştirerek kendi epoch değerlerinize göre modeli tekrar eğitip doğruluk oranlarını kontrol ediniz.
3) Geliştirmiş olduğunuz model ile accuracy, F1-Score ve loss değerlerini control ediniz. Modelin kayıp ve accuracy grafiklerini çizdiriniz.
4) Aynı modeli farklı bir veri seti ile eğitip sonuçlarınızı tartışınız.

Veri Seti Linki:
https://drive.google.com/drive/folders/1EogIAm9YG6XvHDPVlUTTTSk7MZ9LZJZL?usp=share_link