Kaggle Competition Report

<u>Detecting Crop Rows from Image</u>

Introduction

Crop row detection is a critical task in precision agriculture, enabling farmers to optimize their planting, fertilizing, and harvesting operations. The detection of crop rows involves identifying the boundaries of crop rows in an image. In this report, we discuss our experience working on a crop row detection project using deep learning models based on U-Net and LinkNet architectures.

Dataset Description

Each image of train and test images are originally a (320, 240) RGB JPEG image. They are converted to numpy arrays and then flattened to 1D arrays, which contain pixel information. The labels of the training data are contained in the train_labels folder. The test labels are also (320, 240) size images with 3 channels but only contain two types of pixels, 0 and 255. The Run Length Encoding (RLE) method is used for label representation, where the first number is the start index of pixel 255, and the second number is the count.

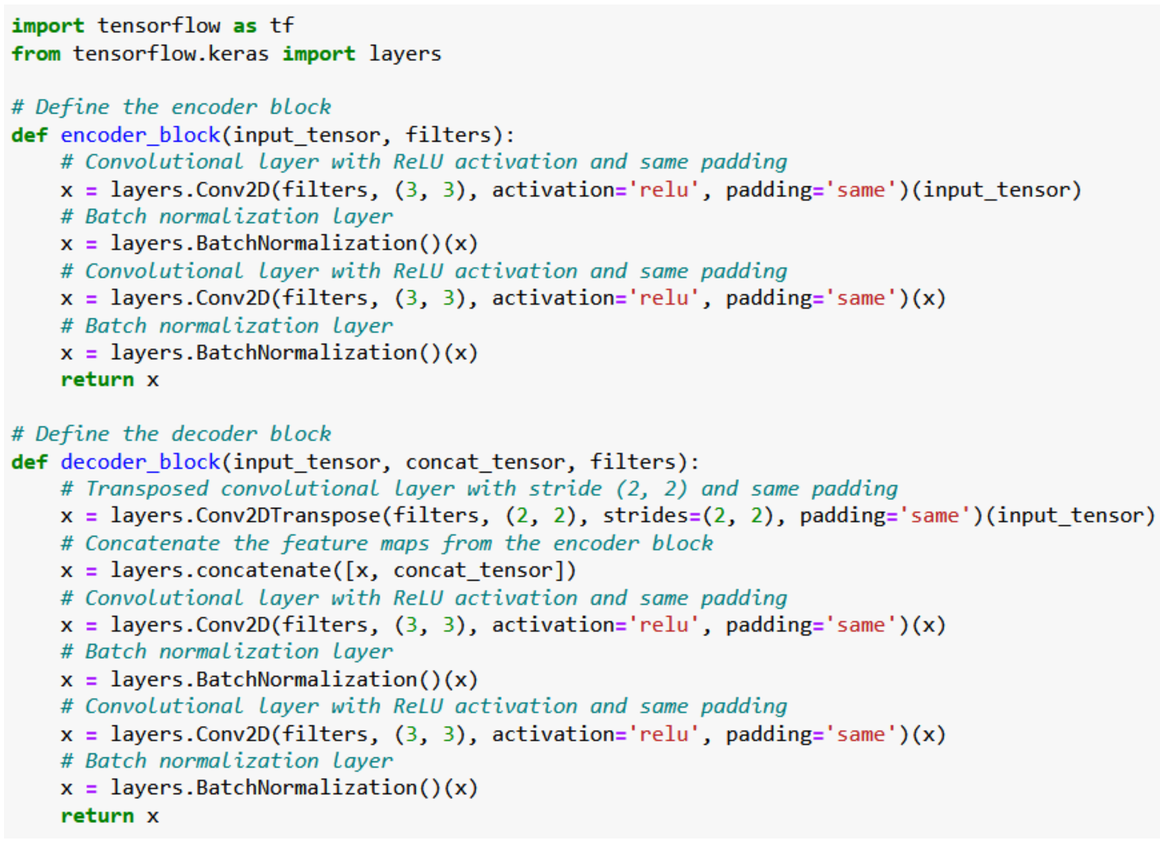
Methodology

To detect crop rows in an image, we trained two models based on U-Net and LinkNet architectures on a dataset of high-resolution satellite images of crop fields. The dataset was labeled with the locations of the crop rows in each image. We preprocessed the dataset by resizing the images to a smaller size to reduce computational requirements. We then split the dataset into training and validation sets and trained our models using the training data. Data augmentation techniques, such as rotation, flipping, and scaling, were applied to increase the models' generalization ability. We also provided functions to encode and decode RLE for the masks.

Data Preprocessing:

The code begins by defining a function ***load_and_preprocess_images()*** to load and preprocess the satellite images. The function takes as input the IDs of the images and the directory where the images are stored, and returns an array of preprocessed images. A similar function ***load_train_labels()*** is defined to load the training labels for the images. The labels are binary masks that indicate the location of the crop rows in each image.

Linknet Architecture:

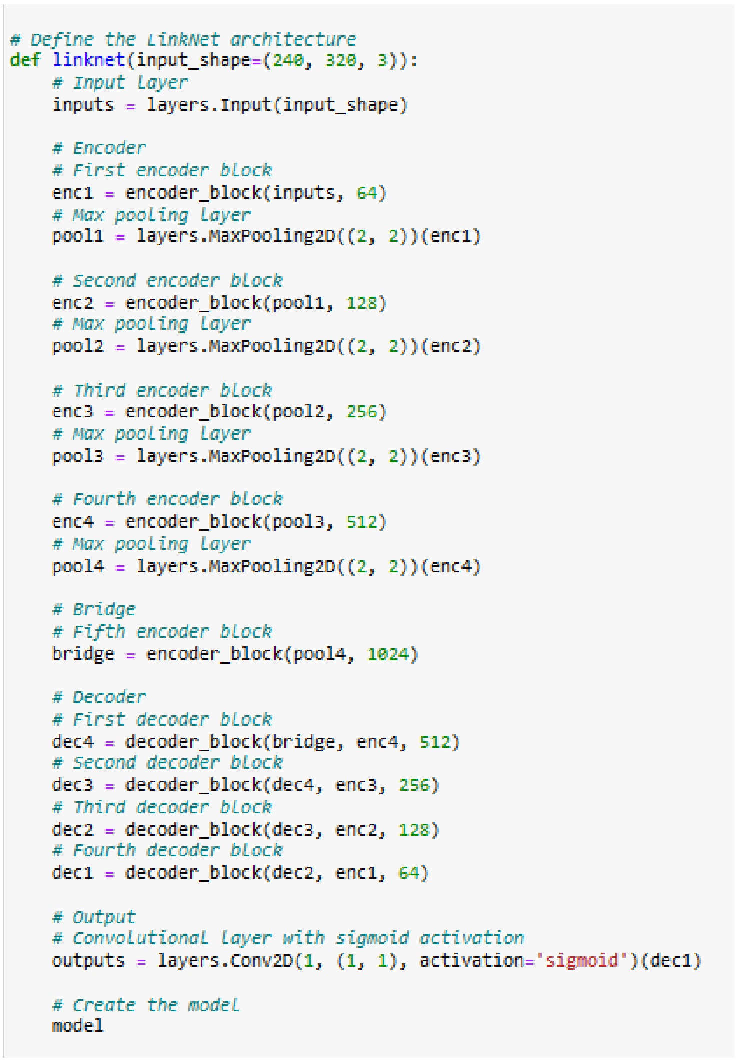

```
import tensorflow as tf
from tensorflow.keras import layers

# Define the encoder block
def encoder_block(input_tensor, filters):
    # Convolutional layer with ReLU activation and same padding
    x = layers.Conv2D(filters, (3, 3), activation='relu', padding='same')(input_tensor)
    # Batch normalization layer
    x = layers.BatchNormalization()(x)
    # Convolutional layer with ReLU activation and same padding
    x = layers.Conv2D(filters, (3, 3), activation='relu', padding='same')(x)
    # Batch normalization layer
    x = layers.BatchNormalization()(x)
    return x

# Define the decoder block
def decoder_block(input_tensor, concat_tensor, filters):
    # Transposed convolutional layer with stride (2, 2) and same padding
    x = layers.Conv2DTranspose(filters, (2, 2), strides=(2, 2), padding='same')(input_tensor)
    # Concatenate the feature maps from the encoder block
    x = layers.concatenate([x, concat_tensor])
    # Convolutional layer with ReLU activation and same padding
    x = layers.Conv2D(filters, (3, 3), activation='relu', padding='same')(x)
    # Batch normalization layer
    x = layers.BatchNormalization()(x)
    # Convolutional layer with ReLU activation and same padding
    x = layers.Conv2D(filters, (3, 3), activation='relu', padding='same')(x)
    # Batch normalization layer
    x = layers.BatchNormalization()(x)
    return x
```

The ***encoder_block*** function takes an input tensor and a number of filters as arguments. It applies a 3x3 convolutional layer with ReLU activation and same padding to the input tensor, followed by a batch normalization layer, and then another 3x3 convolutional layer with ReLU activation and same padding. The output of this function is the result of these operations on the input tensor, which will be used later in the model.

The ***decoder_block*** function performs up-sampling of the input tensor by applying a transposed convolutional layer and concatenating it with the corresponding encoder block tensor. It then applies a series of convolutional layers with ReLU activation and batch normalization to the concatenated tensor. The output is then passed to the next decoder block to increase the spatialresolution of the output.



```
# Define the LinkNet architecture
def linknet(input_shape=(240, 320, 3)):
    # Input layer
    inputs = layers.Input(input_shape)

    # Encoder
    # First encoder block
    enc1 = encoder_block(inputs, 64)
    # Max pooling layer
    pool1 = layers.MaxPooling2D((2, 2))(enc1)

    # Second encoder block
    enc2 = encoder_block(pool1, 128)
    # Max pooling layer
    pool2 = layers.MaxPooling2D((2, 2))(enc2)

    # Third encoder block
    enc3 = encoder_block(pool2, 256)
    # Max pooling layer
    pool3 = layers.MaxPooling2D((2, 2))(enc3)

    # Fourth encoder block
    enc4 = encoder_block(pool3, 512)
    # Max pooling layer
    pool4 = layers.MaxPooling2D((2, 2))(enc4)

    # Bridge
    # Fifth encoder block
    bridge = encoder_block(pool4, 1024)

    # Decoder
    # First decoder block
    dec4 = decoder_block(bridge, enc4, 512)
    # Second decoder block
    dec3 = decoder_block(dec4, enc3, 256)
    # Third decoder block
    dec2 = decoder_block(dec3, enc2, 128)
    # Fourth decoder block
    dec1 = decoder_block(dec2, enc1, 64)

    # Output
    # Convolutional layer with sigmoid activation
    outputs = layers.Conv2D(1, (1, 1), activation='sigmoid')(dec1)

    # Create the model
    model
```

The Linknet model is defined using the ***linknet()*** function. The model consists of an encoder and a decoder. The encoder has four blocks of convolutional and batch normalization layers, followed by max pooling layers.

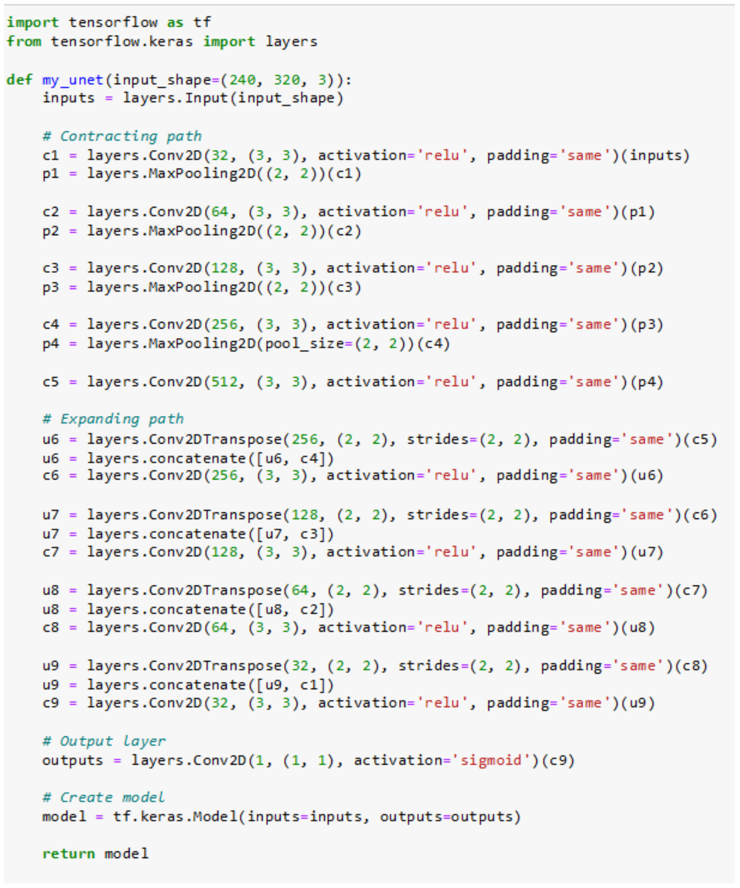
The decoder has four blocks of transposed convolutional layers that upsample the image, concatenated with corresponding feature maps from the encoder, and followed by convolutional and batch normalization layers. The output is a single-channel image with sigmoid activation. The model is then compiled with an Adam optimizer, binary cross-entropy loss, and mean intersection-over-union (IoU) and accuracy metrics. The function takes an optional input shape argument (default: (240, 320, 3)), defines the model architecture, and returns a Keras model object.

The model is then trained using the ***fit()*** function, which trains the model on the training data and evaluates it on the validation data. The training is done for a fixed number of epochs, with a specified batch size and number of steps per epoch.

Finally, the trained model is used to predict the segmentation of a test image. The ***load_and_preprocess_image()*** function loads and preprocesses the image, and the ***predict()*** function of the model is used to generate the segmentation mask. The binary mask is then encoded using run-length encoding, and the results were stored in a dictionary and saved to a CSV file

U-net Architecture:

A U-Net model for image segmentation using the Keras API in TensorFlow. The model consists of a contracting path (encoder) and an expanding path (decoder) with skip connections between them. Each block in the contracting path contains two convolutional layers with ReLU activation followed by max pooling, while each block in the expanding path contains a transposed convolutional layer followed by two convolutional layers with ReLU activation. The output of the model is a binary segmentation mask with one channel.



```
import tensorflow as tf
from tensorflow.keras import layers

def my_unet(input_shape=(240, 320, 3)):
    inputs = layers.Input(input_shape)

    # Contracting path
    c1 = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
    p1 = layers.MaxPooling2D((2, 2))(c1)

    c2 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(p1)
    p2 = layers.MaxPooling2D((2, 2))(c2)

    c3 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(p2)
    p3 = layers.MaxPooling2D((2, 2))(c3)

    c4 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(p3)
    p4 = layers.MaxPooling2D(pool_size=(2, 2))(c4)

    c5 = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(p4)

    # Expanding path
    u6 = layers.Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(c5)
    u6 = layers.concatenate([u6, c4])
    c6 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(u6)

    u7 = layers.Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c6)
    u7 = layers.concatenate([u7, c3])
    c7 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(u7)

    u8 = layers.Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c7)
    u8 = layers.concatenate([u8, c2])
    c8 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(u8)

    u9 = layers.Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same')(c8)
    u9 = layers.concatenate([u9, c1])
    c9 = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(u9)

    # Output layer
    outputs = layers.Conv2D(1, (1, 1), activation='sigmoid')(c9)

    # Create model
    model = tf.keras.Model(inputs=inputs, outputs=outputs)

    return model
```

Results:

Our models were able to detect the boundaries of crop rows in the images. The segmentation results obtained from our models showed clear and precise boundaries of the crop rows, which would be useful for farmers in optimizing their agricultural operations.

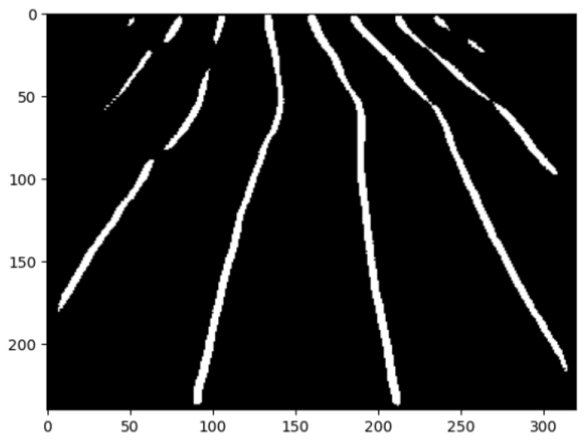


Figure1: Using LinkNet Architecture

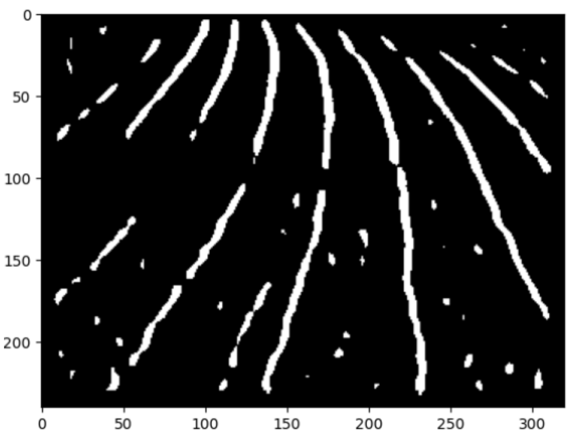


Figure2: Using U-net Architecture

The Intersection over Union (IoU) metric was used to evaluate the models' performance.

Architecture	Mean IoU
Using U-Net Architecture	0.1946081
Using Linknet Architecture	0.24246833

The results suggest that Linknet has a slightly better performance than U-Net on the given task, as it achieved a higher score.

Conclusion:

In conclusion, our experiments with the Link-Net and U-Net architectures for crop row detection show that the Link-Net model outperforms the U-Net model, achieving a higher Intersection over Union (IoU) value of 0.24246833 compared to 0.1946081 for U-Net.

As a participant in this competition, I gained valuable experience in working with deep learning models for computer vision tasks, particularly in the field of precision agriculture. I also learned about the importance of data preprocessing and augmentation techniques for improving model performance.