Land Cover Classification Presentation

Presentation about Land cover Classification

(AI Project presentation)

Project Overview

Mission

The mission of this project is to accurately classify different land cover types, such as forests, croplands, and urban areas, using satellite imagery.

Goal

To develop a fast, robust and reliable machine learning model that can quickly and accurately identify and map different land cover types.

Data & Preprocessing

Data Split

The dataset used was EuroSAT, consists of satellite imagery labeled with 10 different land cover classes, such as deciduous forests, grasslands, and built-up areas.

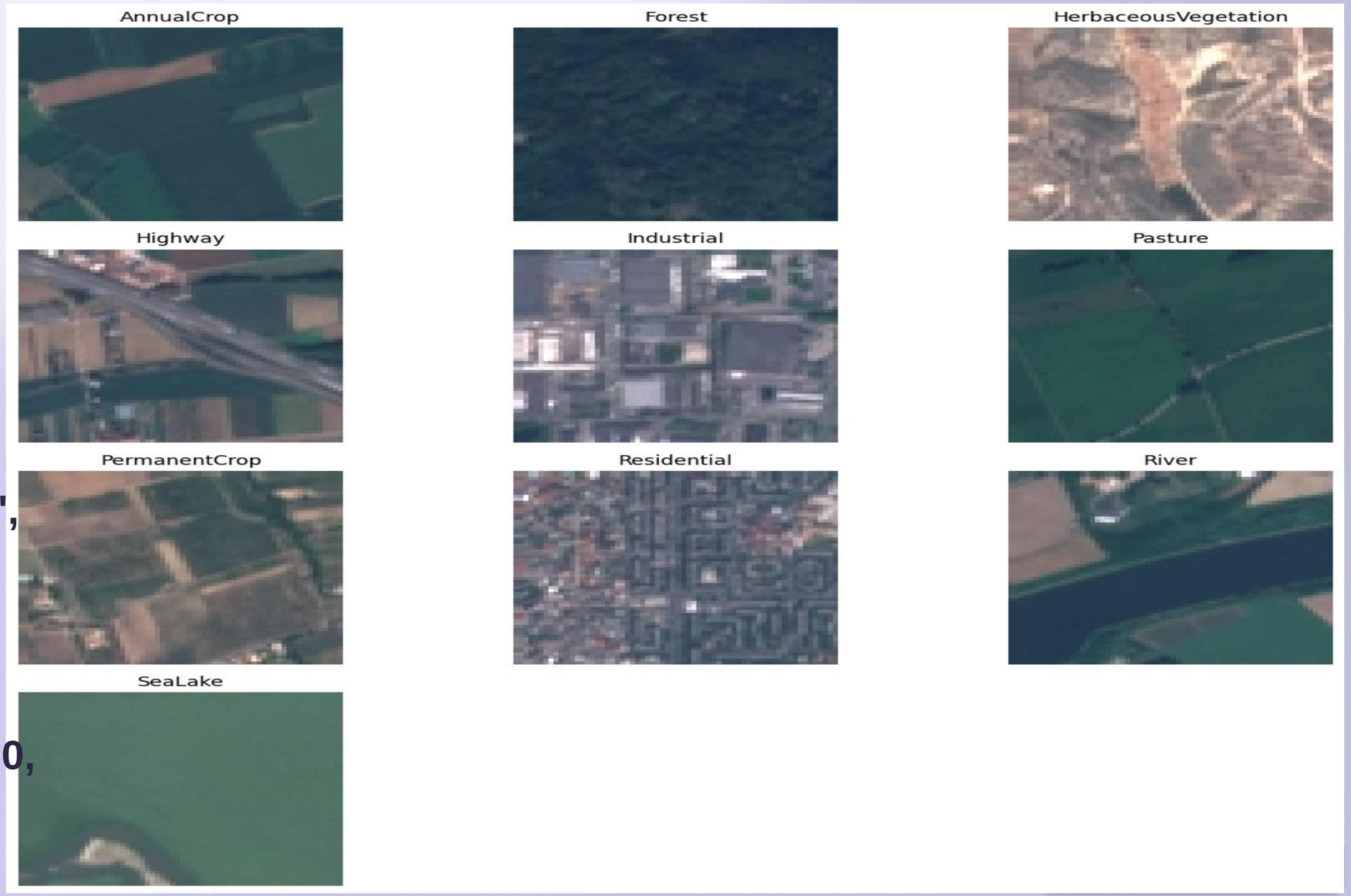
Split:-70% Training || 20% Validation || 10% Testing

Classes: ['AnnualCrop', 'Forest', 'HerbaceousVegetation', 'Highway', 'Industrial', 'Pasture', 'PermanentCrop', 'Residential', 'River', 'SeaLake’]

Number of samples per class: [2400,2400,2400,2000,2000,1600,2000, 2400,2000,2400]

Preprocessing

The images were resized to a common size(64x64) and normalized to improve the model's performance and generalization.



Model Development

1

Baseline Model

The project started with a baseline convolutional neural network (CNN) model, which achieved an initial accuracy of 82%.

2

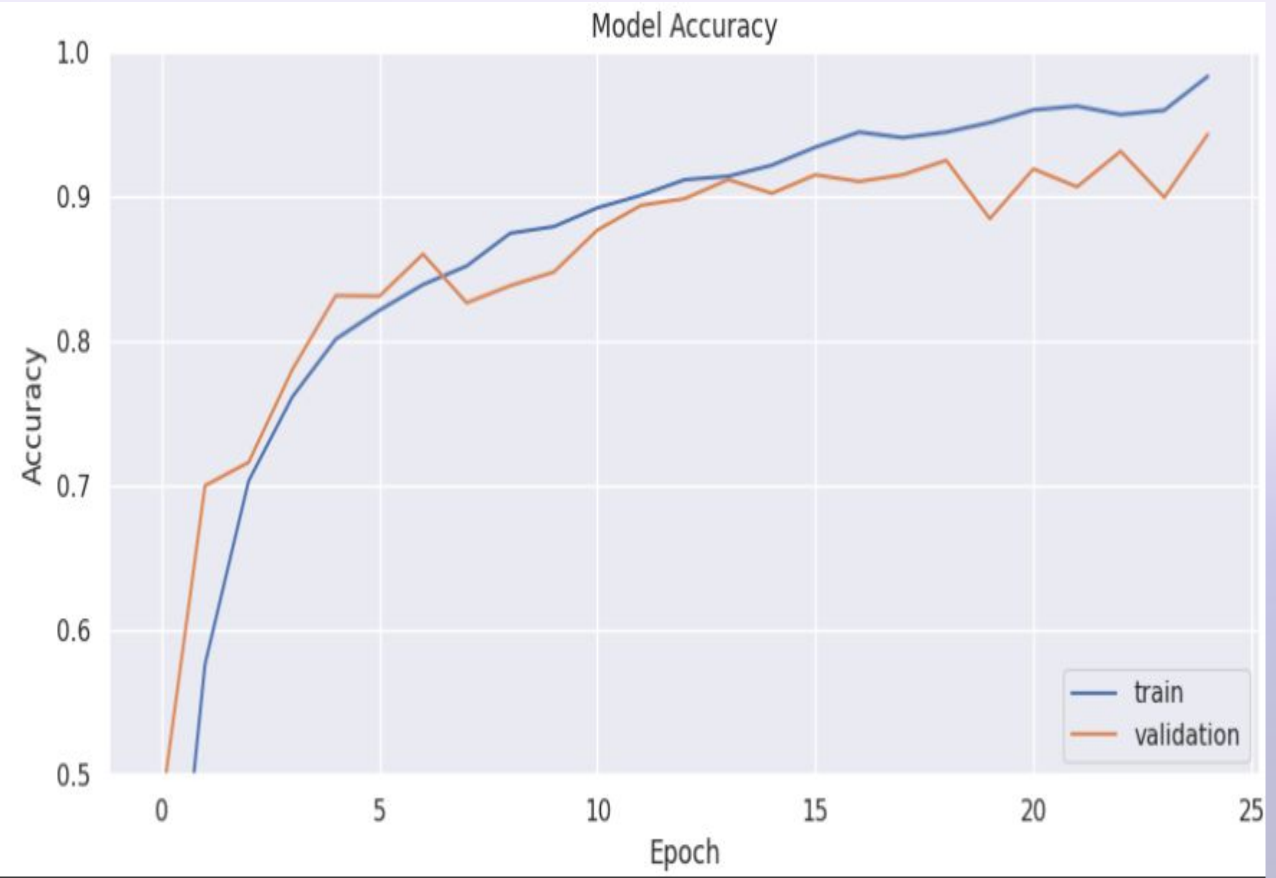
Iterative Improvements

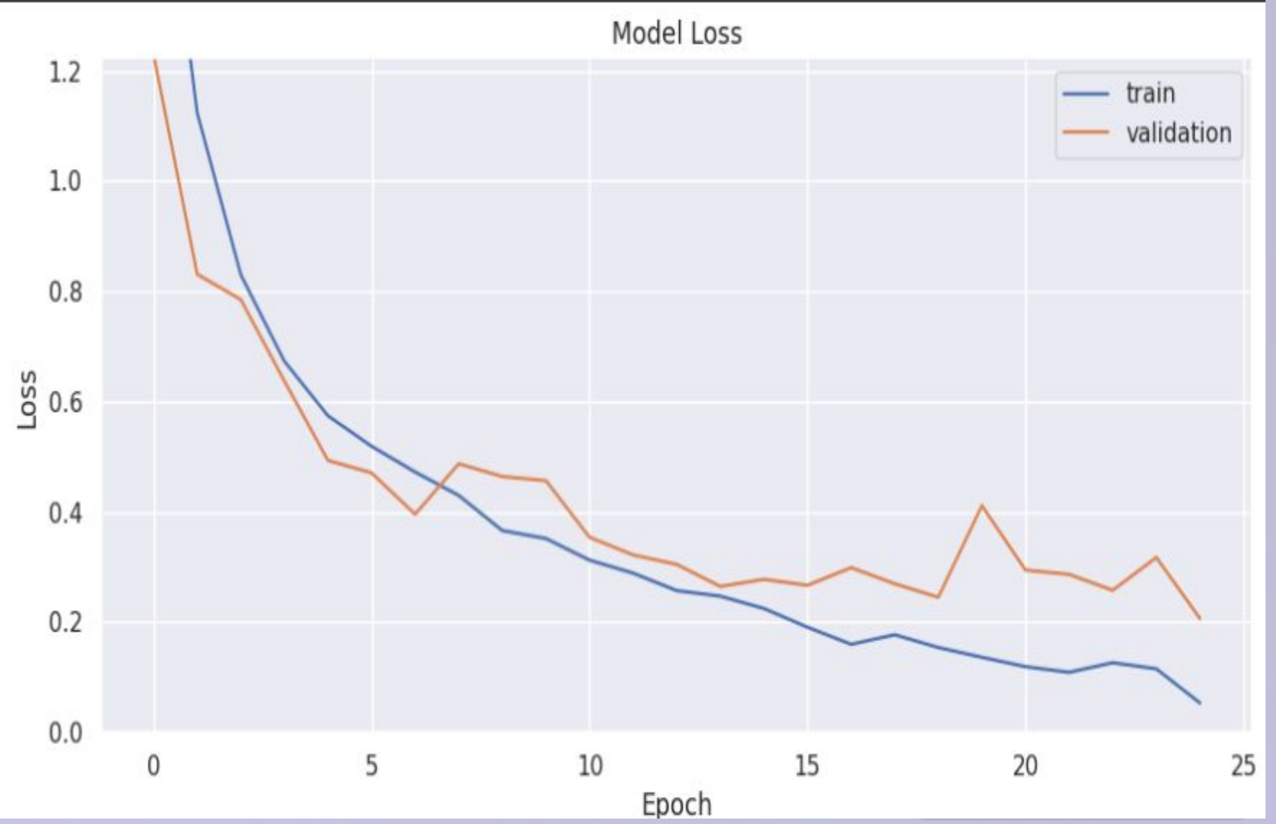
Our team experimented with changing to VGG16, which raised the bar and gave a solid 94%+ accuracy.

3

Transfer Learning and further improvements

We moved ahead with transfer learning approach, applying it over 3 different networks with image-net weights getting close to state-of-art results surpassing 98% mark. Further, We experimented with some adversial attack techniques to test robustness.



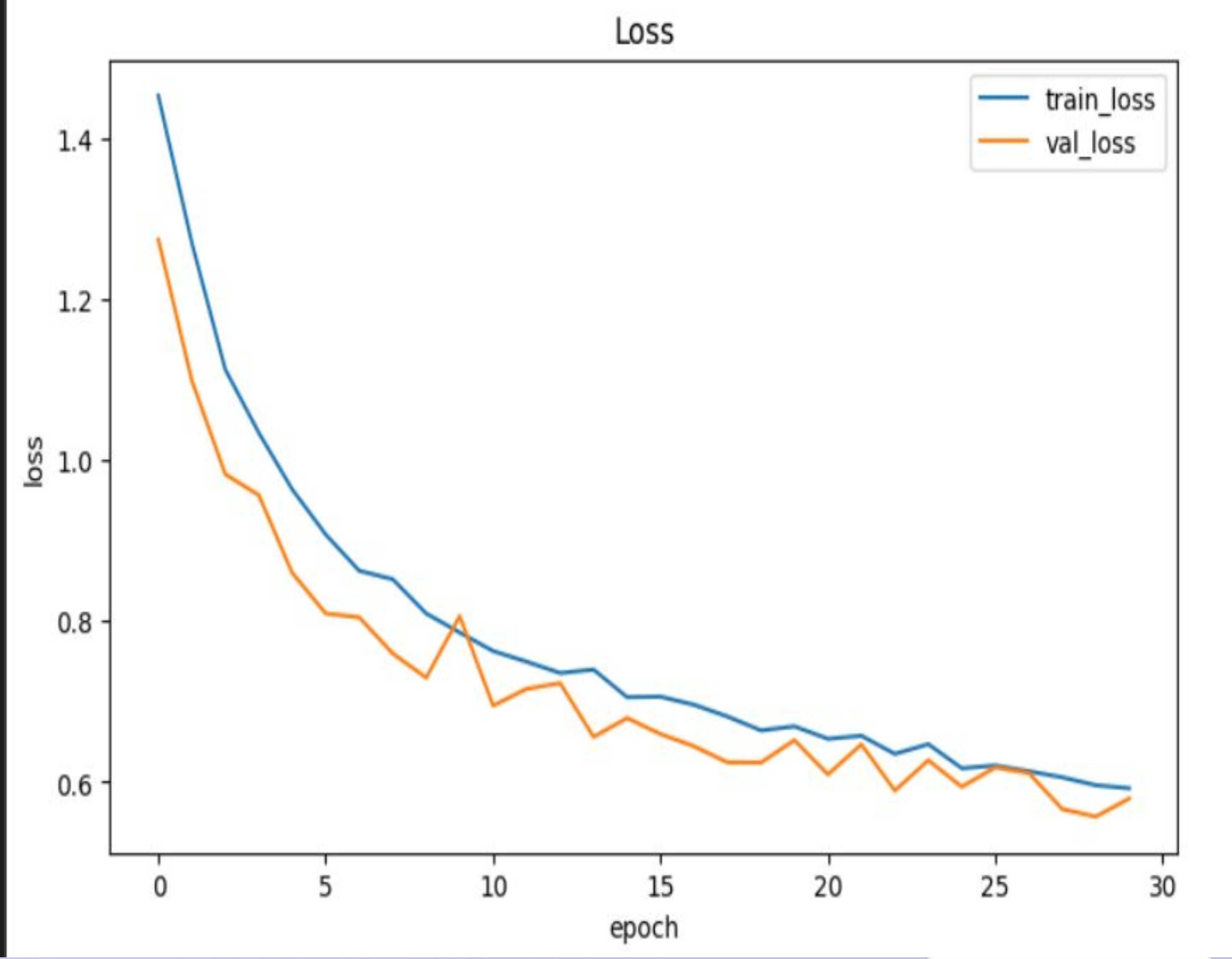


Evaluation & Analysis

Shallow CNN model (Trial-1)(~79%

```
#Really simple and dumb model to begin with lets see how far it takes us
#4 Conv. Layers(32,64,128,128 kernels of 3x3) ->Flatten->Dense->Dropout->Dense(10) cz 10 classes
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)), #rectified linear unit (aka be positive)
    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax') #softmax layer -> e**x/Sigma(e**x)
])
```

```
43/43 [==============================] - 2s 45ms/step - loss: 0.5667 - accuracy: 0.7896
Test Loss: 0.5667493343353271
Test Accuracy: 0.7896296381950378
```

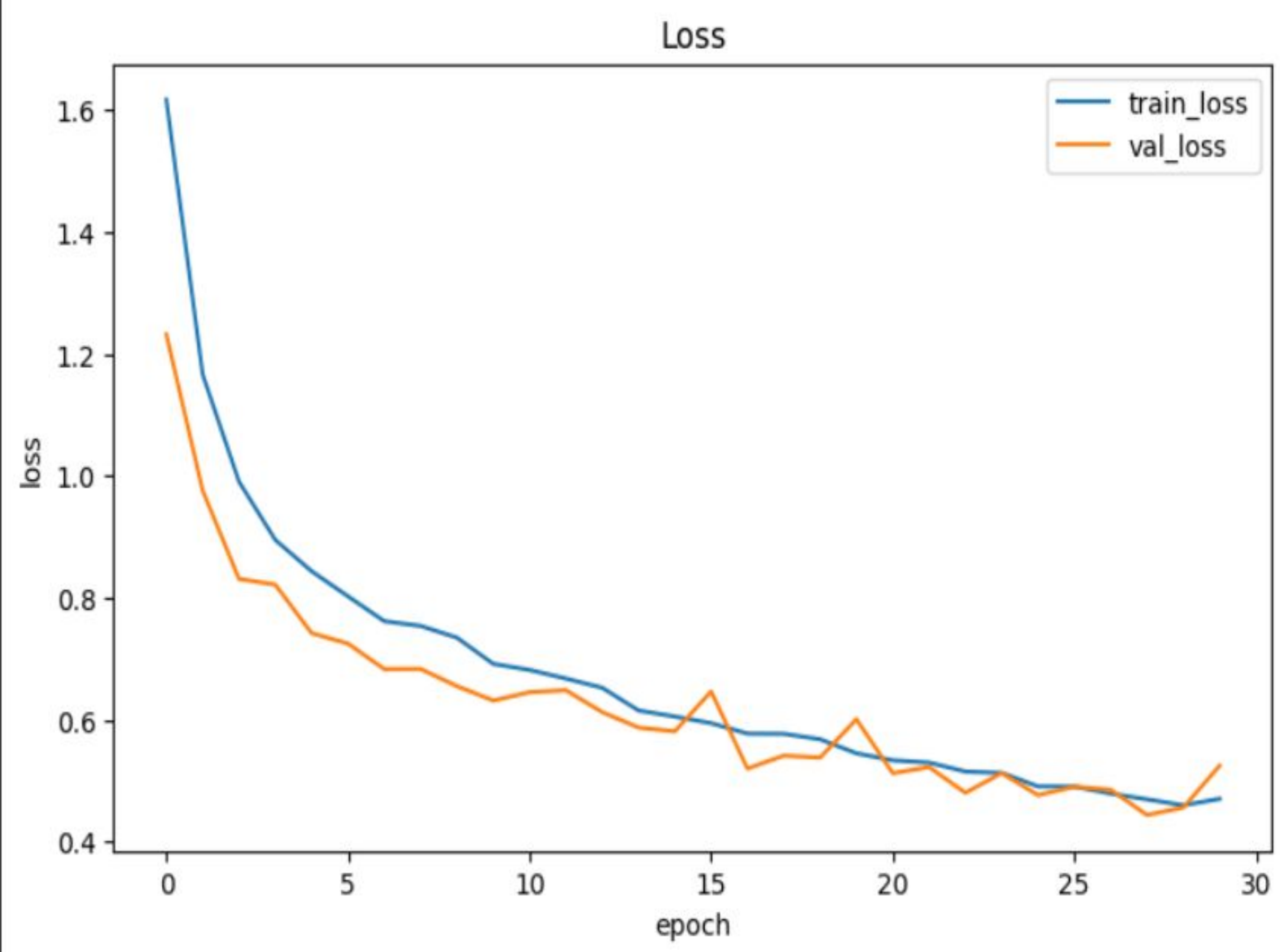


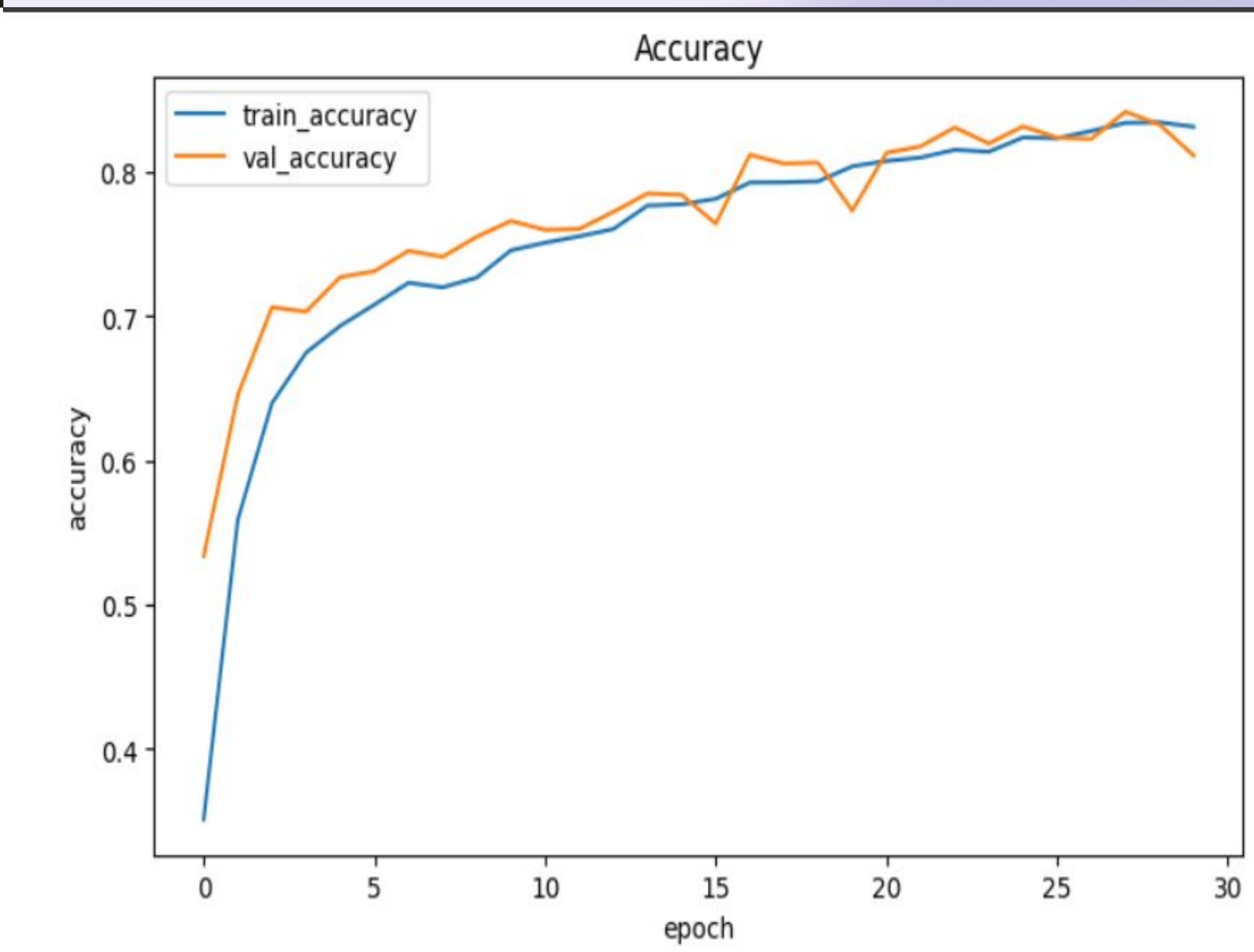
Evaluation & Analysis

Shallow CNN model (Trial-2)(~82% accuracy)

Batch-size reduced from 64 to 32

```
#making everything equal!!!!!!
image_height, image_width = 64, 64
batch_size = 32 # reduced batch size
train_generator = ImageDataGenerator(rescale=1./255, validation_split=0.2)
```





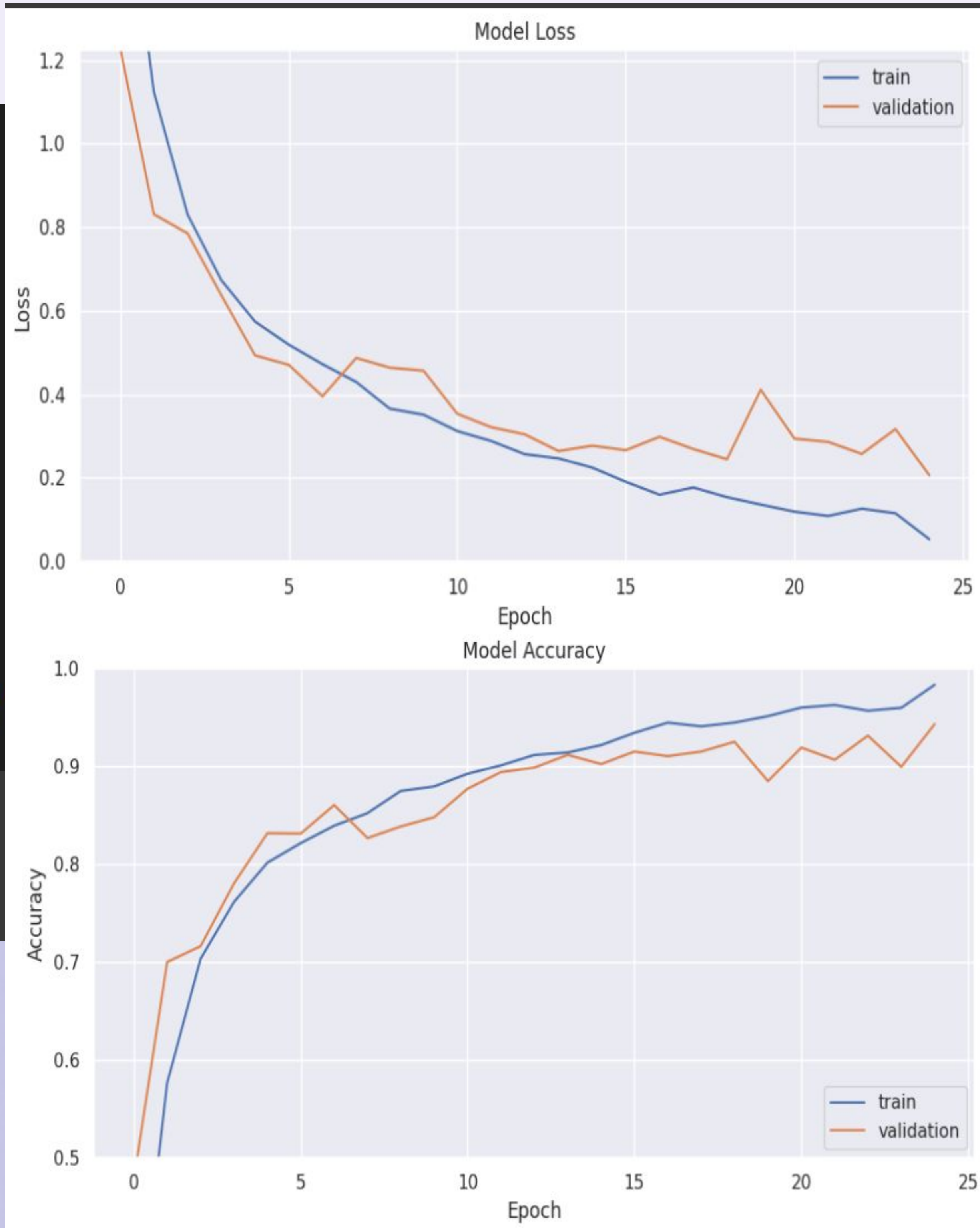
Evaluation & Analysis

VGG16 without pre-training (94%+

```
#Say Hello to VGG-Visual Geometry Group, A CNN for simple and effective image classification developed in 2014 by Oxford
base_vgg16 = VGG16(include_top=False,
                   weights=None, #lets go random!!!!!!!
                   input_tensor=None,
                   input_shape=(64, 64, 3),
                   pooling=None,
                   classes=10,
                   classifier_activation='softmax')
```

```
model_path = "/content/eurosat_rgb_vgg16_model_no_weights.h5"
checkpoint = ModelCheckpoint(filepath=model_path, monitor="val_loss", save_best_only=True)
reduce_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=5)
early_stopping = EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True, verbose=True)
callback_list = [checkpoint, early_stopping, reduce_lr]
```

```
43/43 [==============================] - 1s 31ms/step - loss: 0.1978 - accuracy: 0.9433
Test Loss: 0.19777628779411316
Test Accuracy: 0.9433333277702332
```



<u>**13 Conv Layers 3 Fully connected layers in VGG16**</u>

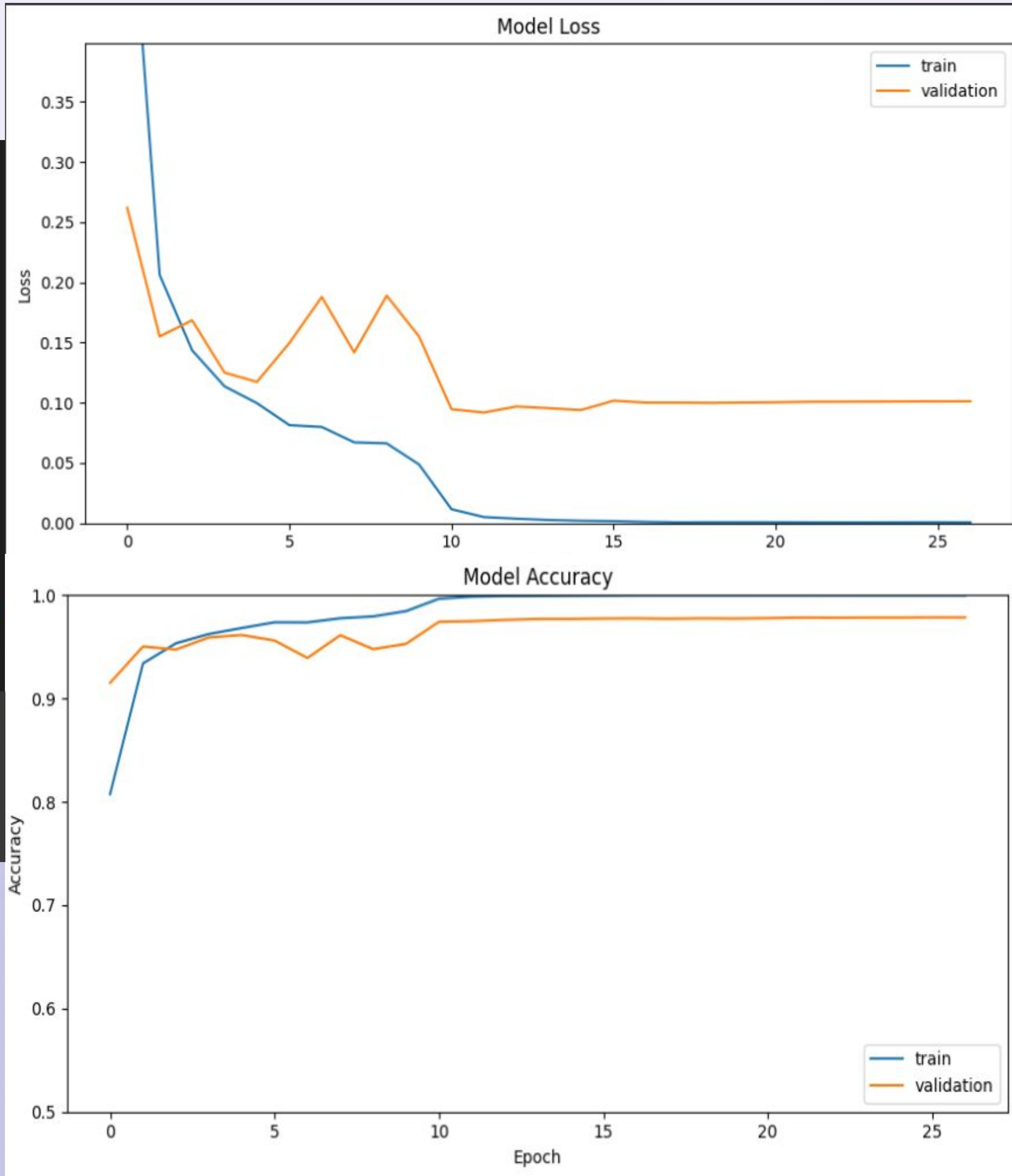
Evaluation & Analysis

VGG16 with pre-training (98%+ accuracy)

```
base_vgg16 = VGG16(include_top=False,
                   weights='imagenet',
                   input_tensor=None,
                   input_shape=(64, 64, 3),
                   pooling=None,
                   classes=10,
                   classifier_activation='softmax')
```

```
model_path = "/content/eurosat_rgb_vgg16_model_no_weights.h5"
checkpoint = ModelCheckpoint(filepath=model_path, monitor="val_loss", save_best_only=True)
reduce_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=5)
early_stopping = EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True, verbose=True)
callback_list = [checkpoint, early_stopping, reduce_lr]
```

```
43/43 [==============================] - 2s 34ms/step - loss: 0.0752 - accuracy: 0.9807
Test Loss: 0.0751752182841301
Test Accuracy: 0.9807407259941101
```



<u>Possibility of overfitting having occurred, can not be absolutely sure due to 98% accuracy but a better model may help in much more robust results, we will dwell into the territory of robustness too for a short period later.</u>

Evaluation & Analysis

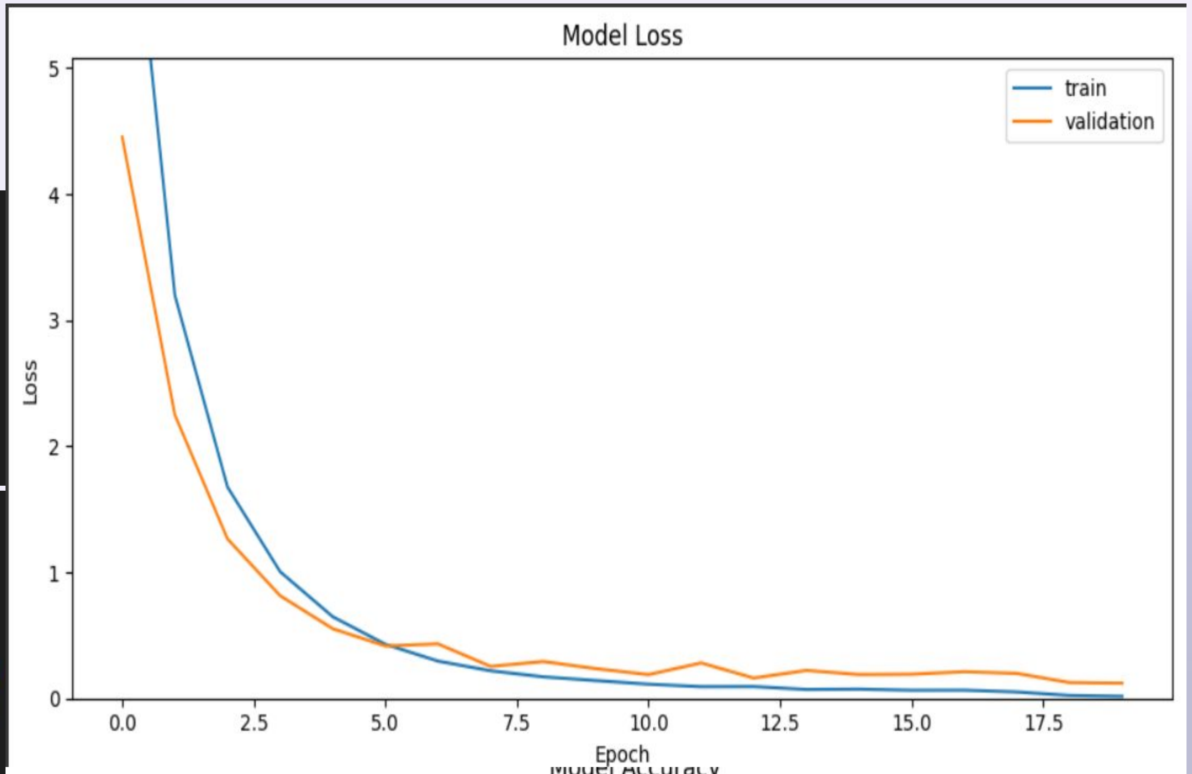
VGG16 with pre-training (98%+ accuracy)

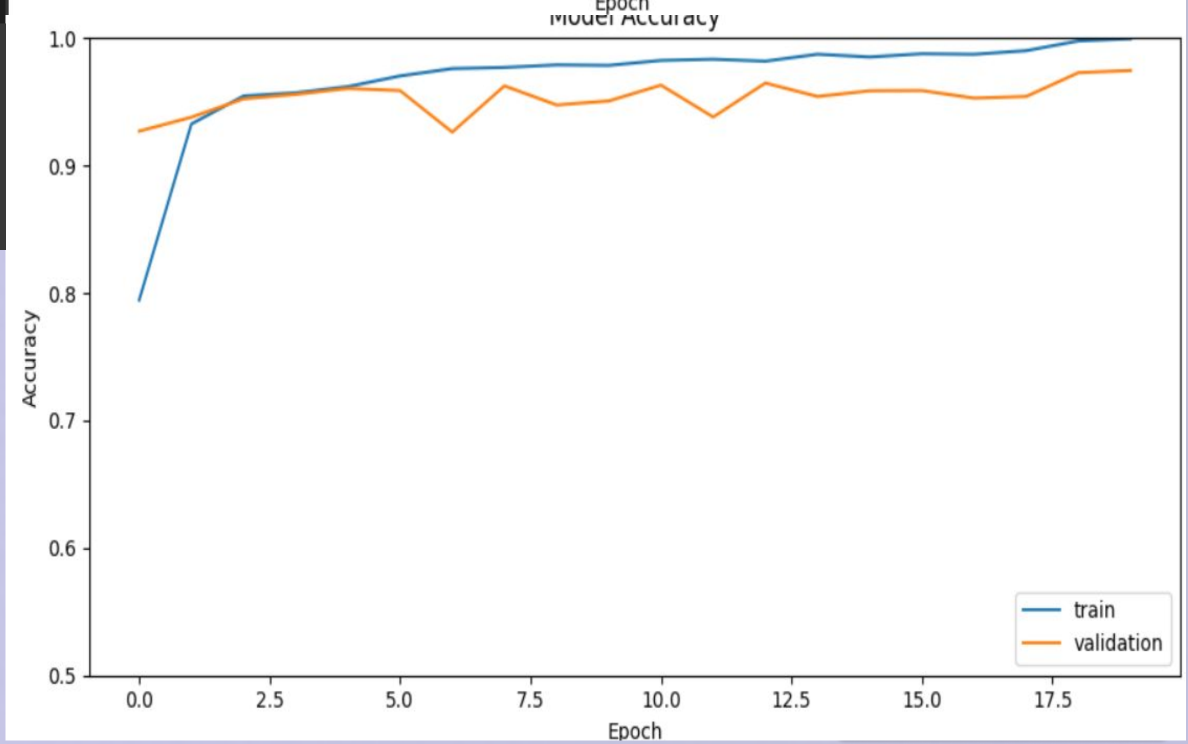
Use of L2 Regularization(Ridge regularization)

```python
# L2 regularization
model.add(Dense(units=512, activation="relu",
                kernel_initializer="he_normal",
                kernel_regularizer=l2(0.01)))  # L2 regularization with weight decay
```

```python
model_path = "/content/eurosat_rgb_vgg16_model_no_weights.h5"
checkpoint = ModelCheckpoint(filepath=model_path, monitor="val_loss", save_best_only=True)
reduce_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=5)
early_stopping = EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True, verbose=True)
callback_list = [checkpoint, early_stopping, reduce_lr]
```

```
43/43 [==============================] - 2s 35ms/step - loss: 0.1041 - accuracy: 0.9800
Test Loss: 0.10413076728582382
Test Accuracy: 0.9800000190734863
```





<u>**No major improvement in accuracy but we will shortly see a major advantage of applying regularizers.**</u>

Evaluation & Analysis

VGG19 with pre-training (98.25%+)

```
#Let us try VGG19, a slightly more advanced model
base_vgg19 = VGG19(include_top=False,
                   weights='imagenet',
                   input_tensor=None,
                   input_shape=(64, 64, 3),
                   pooling=None,
                   classes=10,
                   classifier_activation='softmax')
```

```
43/43 [==============================] - 2s 55ms/step - loss: 0.0698 - accuracy: 0.9826
Test Loss: 0.06977824121713638
Test Accuracy: 0.9825925827026367
```

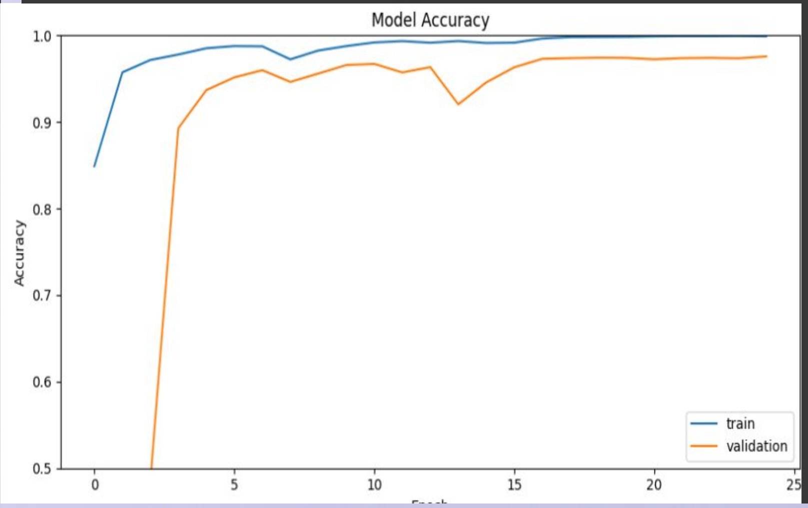
ResNet50 with pre-training(~98.4%)

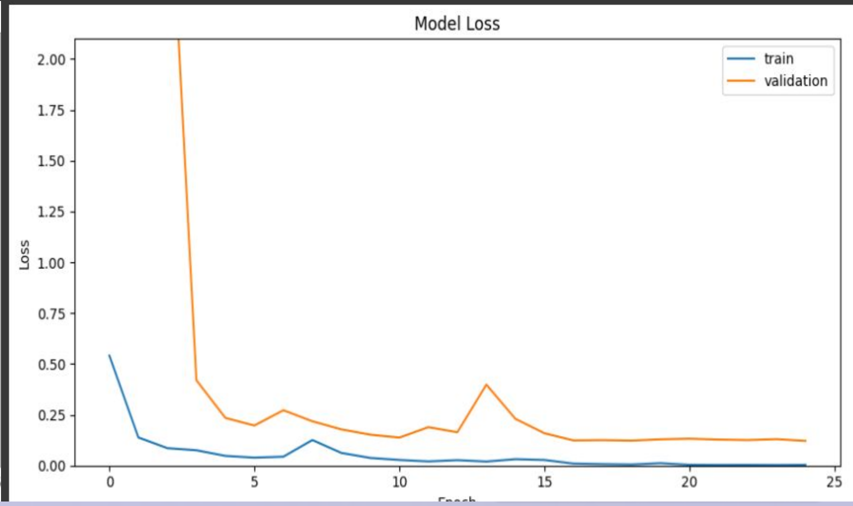
```
base_resnet50 = ResNet50(
                include_top=False,
                weights="imagenet",
                input_tensor=None,
                input_shape=(64, 64, 3),
                pooling=None,
                classes=10,
                classifier_activation='softmax',
        )
```

```
43/43 [==============================] - 2s 54ms/step - loss: 0.0890 - accuracy: 0.9837
Test Loss: 0.08902817219495773
Test Accuracy: 0.9837037324905396
```

We were able to achieve near state of art performance with VGG19 and VGG16, as well as confirmed the paper’s finding about ResNet50 by achieving accuracy around the benchmark levels.

Goes on to show that VGG16 and VGG19 models are able to reach state-of-art performances despite being much shallower.





Evaluation & Analysis

Adversial attacks

AugMix-Like adversial data

```python
augmenter = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode="nearest"
)

# Apply augmentation to the test images
augmented_images = np.array([augmenter.random_transform(img) for img in test_images])
```

```
786/786 [==============================] - 16s 19ms/step - loss: 0.2107 - accuracy: 0.9462
Test Loss with Augmented Images: 0.2107182741165161
Test Accuracy with Augmented Images: 0.9461642503738403
```

Not a large drop on VGG19(non-regularized version): ~95% accuracy.

So, We worked upon more advanced adversial attacking methods.

FGSM adversial attack method

```python
test_images, test_labels = np.array(test_images), np.array(test_labels)
perturbed_images = []
for img in test_images:
    img_tensor = tf.convert_to_tensor(img.reshape(1, *img.shape))
    with tf.GradientTape() as tape:
        tape.watch(img_tensor)
        prediction = model(img_tensor)
        loss = tf.keras.losses.categorical_crossentropy(test_labels[0].reshape(1, -1), prediction)
    gradient = tape.gradient(loss, img_tensor)
```

Effect on vanilla VGG19(~81% only)

```
392/392 [==========================] - 8s 21ms/step - loss: 1.5373 - accuracy: 0.8155
Test Loss with FGSM-perturbed Images: 1.537313461303711
Test Accuracy with FGSM-perturbed Images: 0.815470516816711
```

Evaluation & Analysis

Adversial attacks

FGSM adversial attack method

Effect on vanilla ResNet(~84% accuracy)

```
197/197 [==============================] - 3s 17ms/step - loss: 3.9149 - accuracy: 0.8437
Test Loss with FGSM-perturbed Images: 3.9149274826049805
Test Accuracy with FGSM-perturbed Images: 0.8437103033065796
```

Still a Loss of around 15% for untargeted attacks along with a large loss.

Improvement on robustness due to more trainable parameters/layers.

Effect on regularized VGG16(~87%)

```
392/392 [==============================] - 6s 16ms/step - loss: 1.5367 - accuracy: 0.8683
Test Loss with FGSM-perturbed Images: 1.536657691001892
Test Accuracy with FGSM-perturbed Images: 0.868341326713562
```

Good statistics keeping in consideration the number of trainable parameters. L2 regularization makes model much more robust.

Conclusion

This project successfully showed VGG16 model's ability to provide a near- state-of-art and robust enough solution to the problem of classification of land through satellite imagery, through EuroSAT dataset. The land type detection can find various use cases in field of technology, agriculture and city planning.