Name: Soumya Ranjan Nayak

PRN: 23070243063

Assignment: Deep Learning: CNN

```
import kagglehub
puneet6060_intel_image_classification_path = kagglehub.dataset_download('puneet6060/intel-image-classification')

print('Data source import complete.')
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import os
import glob as gb
import cv2
import keras
from tensorflow.keras.models import Sequential, Model
```

```
trainpath = '/kaggle/input/intel-image-classification/seg_train/seg_train'
testpath = '/kaggle/input/intel-image-classification/seg_test/seg_test'
predpath = '/kaggle/input/intel-image-classification/seg_pred/seg_pred'
```

```
IMAGE_SIZE = (228, 228)

BATCH_SIZE = 32
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode="nearest"
)

train_ds = datagen.flow_from_directory(
    trainpath,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='sparse'
)
```

```
Found 14034 images belonging to 6 classes.
```

```
test_ds = tf.keras.utils.image_dataset_from_directory(
    testpath,
    seed=123,
    image_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE
)
```

```
Found 3000 files belonging to 6 classes.
```

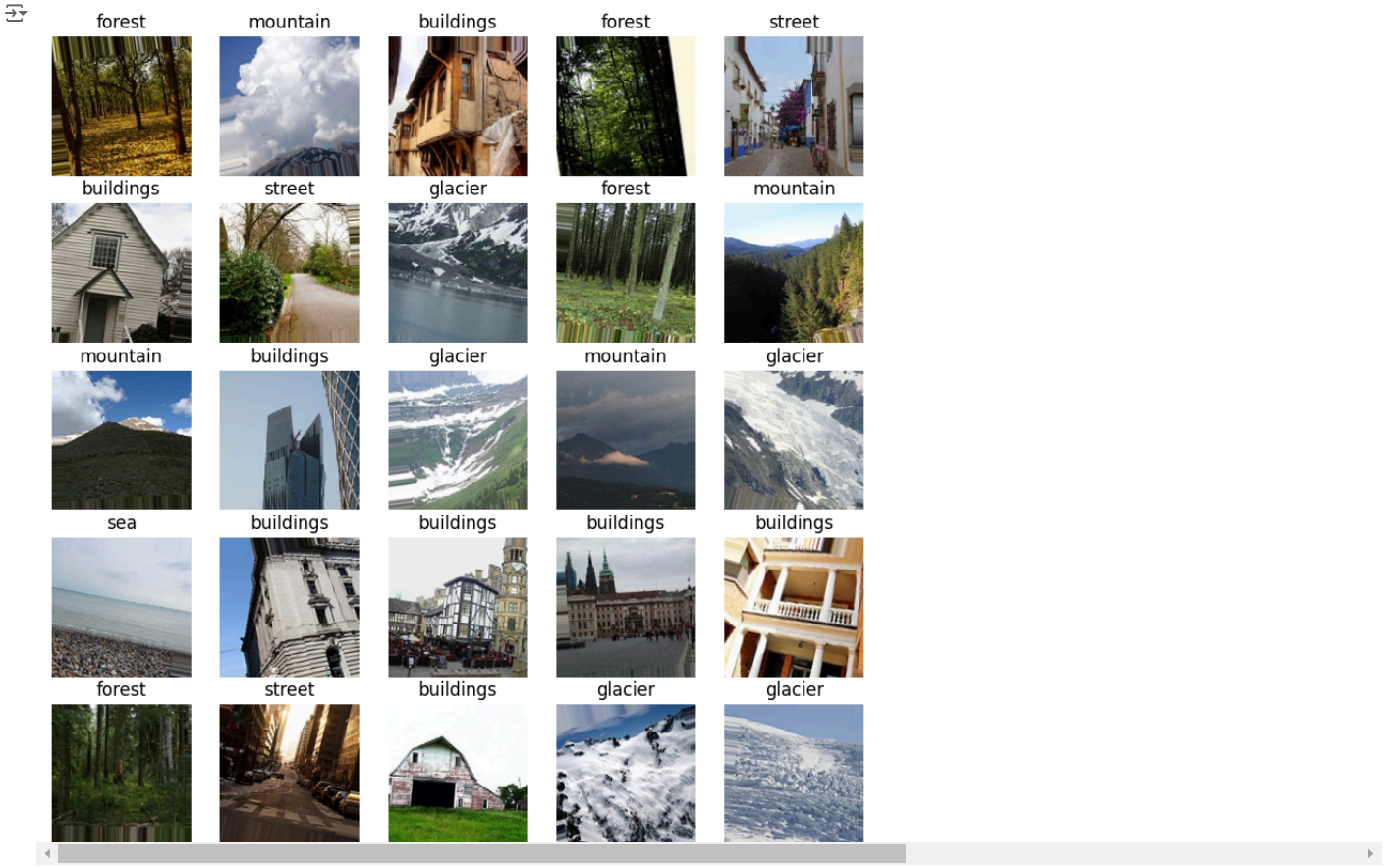
```
class_names = list(train_ds.class_indices.keys())
print(class_names)
```

```
['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']
```

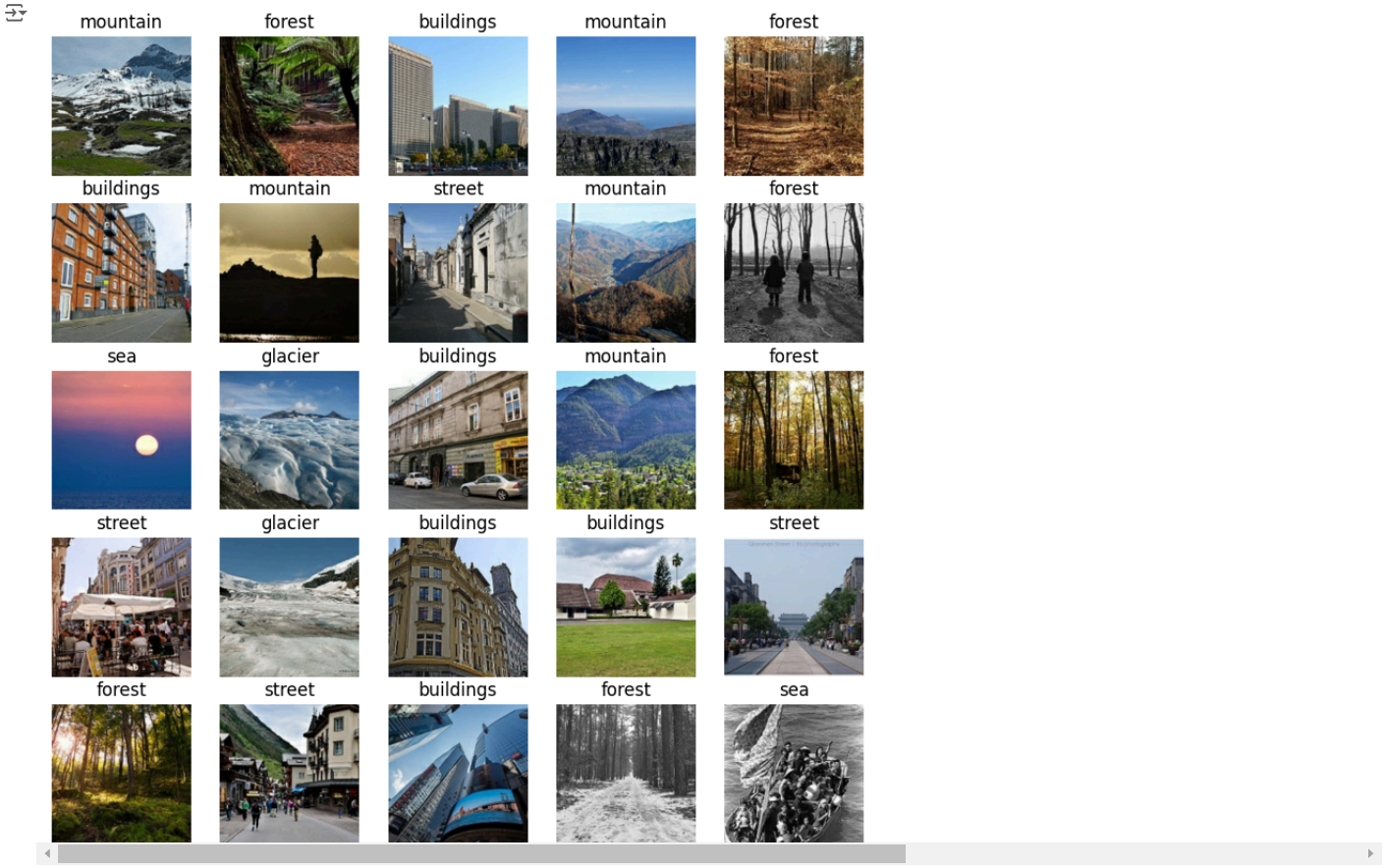
```
def getImagePaths(path):
    image_names = []
    for dirname, _, filenames in os.walk(path):
        for filename in filenames:
            fullpath = os.path.join(dirname, filename)
            image_names.append(fullpath)
    return image_names
images_paths = getImagePaths(predpath)
len(images_paths)
```

```
7301
```

```
plt.figure(figsize=(10, 10))

# Fetch a batch of images and labels
images, labels = next(train_ds)

# Display 25 images from the batch
for i in range(25):
    ax = plt.subplot(5, 5, i + 1)
    plt.imshow(images[i].astype("uint8"))
    plt.title(class_names[int(labels[i])])
    plt.axis("off")

plt.show()
```




```
plt.figure(figsize=(10, 10))
for images, labels in test_ds.take(1):
    for i in range(25):
        ax = plt.subplot(5, 5, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```




```
import tensorflow.keras.models as Models
```

```
model = Models.Sequential()
model.add(tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(228, 228, 3)))
model.add(tf.keras.layers.BatchNormalization())
```

```
model.add(tf.keras.layers.MaxPooling2D(2, 2))

model.add(tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(2, 2))

model.add(tf.keras.layers.Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(2, 2))

model.add(tf.keras.layers.Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(2, 2))

model.add(tf.keras.layers.Conv2D(256, kernel_size=(3, 3), activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(2, 2))

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(1024, activation='relu'))
model.add(tf.keras.layers.Dropout(0.15))
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dropout(0.1))
model.add(tf.keras.layers.Dense(len(class_names), activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)          (None, 226, 226, 32)      896

 batch_normalization_15 (Ba  (None, 226, 226, 32)      128
 tchNormalization)

 max_pooling2d_15 (MaxPooli  (None, 113, 113, 32)      0
 ng2D)

 conv2d_16 (Conv2D)          (None, 111, 111, 64)      18496

 batch_normalization_16 (Ba  (None, 111, 111, 64)      256
 tchNormalization)

 max_pooling2d_16 (MaxPooli  (None, 55, 55, 64)        0
 ng2D)

 conv2d_17 (Conv2D)          (None, 53, 53, 128)       73856

 batch_normalization_17 (Ba  (None, 53, 53, 128)       512
 tchNormalization)

 max_pooling2d_17 (MaxPooli  (None, 26, 26, 128)       0
 ng2D)

 conv2d_18 (Conv2D)          (None, 24, 24, 128)       147584

 batch_normalization_18 (Ba  (None, 24, 24, 128)       512
 tchNormalization)

 max_pooling2d_18 (MaxPooli  (None, 12, 12, 128)       0
 ng2D)

 conv2d_19 (Conv2D)          (None, 10, 10, 256)       295168

 batch_normalization_19 (Ba  (None, 10, 10, 256)       1024
 tchNormalization)

 max_pooling2d_19 (MaxPooli  (None, 5, 5, 256)         0
 ng2D)

 flatten_3 (Flatten)         (None, 6400)              0

 dense_9 (Dense)             (None, 1024)              6554624

 dropout_6 (Dropout)         (None, 1024)              0

 dense_10 (Dense)            (None, 256)               262400

 dropout_7 (Dropout)         (None, 256)               0

 dense_11 (Dense)            (None, 6)                 1542

=================================================================
Total params: 7356998 (28.06 MB)
```

```
from tensorflow.keras.optimizers import Adam
model.compile(
    optimizer = Adam(learning_rate = 0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    #loss = "categorical_crossentropy",
    metrics = ["accuracy"])
```

```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

earlystopping = EarlyStopping(monitor='val_loss',
                              patience=5,
                              verbose=1,
                              mode='min'
                             )

checkpointer = ModelCheckpoint(filepath='bestvalue.keras', verbose=0, save_best_only=True)
callback_list = [checkpointer, earlystopping]
```

```
#converting into dictionaries
from sklearn.utils.class_weight import compute_class_weight
import os
import numpy as np

class_names = sorted(os.listdir(trainpath))
label_counts = []

for class_name in class_names:
    class_dir = os.path.join(trainpath, class_name)
    label_counts.append(len(os.listdir(class_dir)))

all_labels = np.concatenate([[i] * count for i, count in enumerate(label_counts)])


class_weights_array = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(all_labels),
    y=all_labels
)

class_weights = {i: weight for i, weight in enumerate(class_weights_array)}

print(class_weights)
```

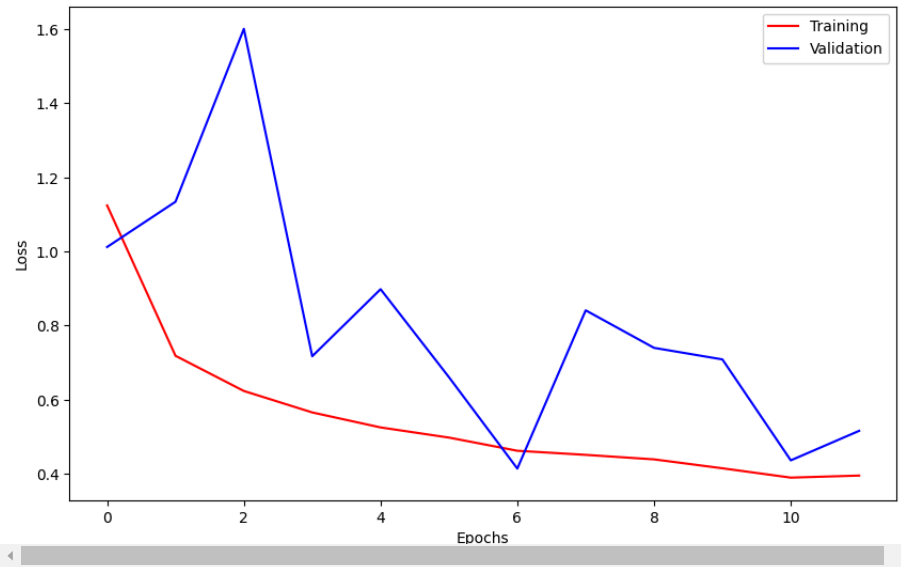
```
{0: 1.067549064354176, 1: 1.0299427564949362, 2: 0.9729617304492513, 3: 0.9311305732484076, 4: 1.0285839929639402, 5: 0.9819479429051218}
```

```
history = model.fit(
    train_ds,
    validation_data=test_ds,
    epochs=40,
    class_weight=class_weights,
    callbacks=callback_list
)
```

```
Epoch 1/40
439/439 [==============================] - 254s 571ms/step - loss: 1.1242 - accuracy: 0.6216 - val_loss: 1.0121 - val_accuracy: 0.6237
Epoch 2/40
439/439 [==============================] - 245s 559ms/step - loss: 0.7184 - accuracy: 0.7419 - val_loss: 1.1342 - val_accuracy: 0.5960
Epoch 3/40
439/439 [==============================] - 247s 562ms/step - loss: 0.6234 - accuracy: 0.7783 - val_loss: 1.6009 - val_accuracy: 0.4990
Epoch 4/40
439/439 [==============================] - 245s 558ms/step - loss: 0.5655 - accuracy: 0.8013 - val_loss: 0.7171 - val_accuracy: 0.7657
Epoch 5/40
439/439 [==============================] - 251s 572ms/step - loss: 0.5250 - accuracy: 0.8140 - val_loss: 0.8981 - val_accuracy: 0.7250
Epoch 6/40
439/439 [==============================] - 250s 570ms/step - loss: 0.4976 - accuracy: 0.8240 - val_loss: 0.6606 - val_accuracy: 0.7603
Epoch 7/40
439/439 [==============================] - 231s 526ms/step - loss: 0.4621 - accuracy: 0.8336 - val_loss: 0.4140 - val_accuracy: 0.8580
Epoch 8/40
439/439 [==============================] - 238s 543ms/step - loss: 0.4510 - accuracy: 0.8426 - val_loss: 0.8411 - val_accuracy: 0.6703
Epoch 9/40
439/439 [==============================] - 226s 514ms/step - loss: 0.4388 - accuracy: 0.8499 - val_loss: 0.7398 - val_accuracy: 0.7587
Epoch 10/40
439/439 [==============================] - 209s 475ms/step - loss: 0.4147 - accuracy: 0.8546 - val_loss: 0.7086 - val_accuracy: 0.7723
Epoch 11/40
439/439 [==============================] - 210s 477ms/step - loss: 0.3895 - accuracy: 0.8635 - val_loss: 0.4361 - val_accuracy: 0.8513
Epoch 12/40
439/439 [==============================] - 217s 493ms/step - loss: 0.3951 - accuracy: 0.8625 - val_loss: 0.5158 - val_accuracy: 0.8263
Epoch 12: early stopping
```

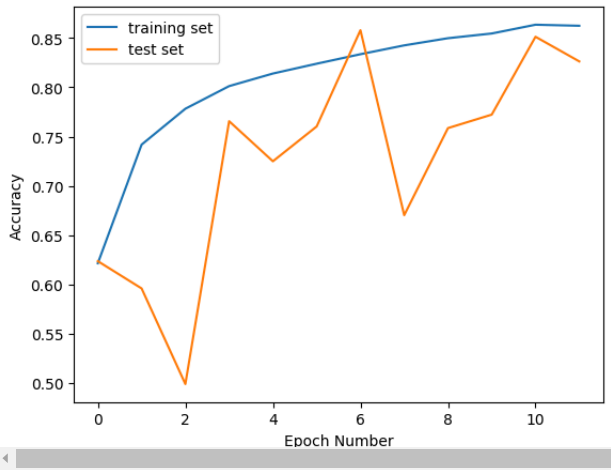
```
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(loss))

fig = plt.figure(figsize=(10,6))
plt.plot(epochs,loss,c="red",label="Training")
plt.plot(epochs,val_loss,c="blue",label="Validation")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7cd9f81b8490>
```



```
plt.xlabel('Epoch Number')
plt.ylabel('Accuracy')
plt.plot(history.history['accuracy'], label='training set')
plt.plot(history.history['val_accuracy'], label='test set')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7cd9f820df30>
```



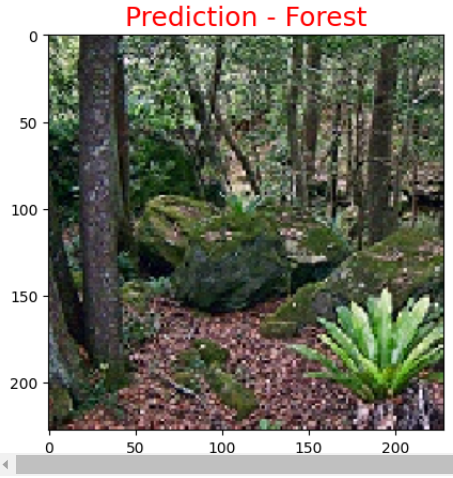

```
def predict_image(filename, model):
    img_ = image.load_img(filename, target_size=(228, 228))
    img_array = image.img_to_array(img_)
    img_processed = np.expand_dims(img_array, axis=0)
    img_processed /= 255.

    prediction = model.predict(img_processed)

    index = np.argmax(prediction)

    plt.title("Prediction - {}".format(str(class_names[index]).title()), size=18, color='red')
    plt.imshow(img_array)
```

```
test_loss, test_accuracy = model.evaluate(test_ds)

print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
94/94 [==============================] - 3s 25ms/step - loss: 0.5158 - accuracy: 0.8263
Test Accuracy: 82.63%
```

```
#testing with sample data
from tensorflow.keras.preprocessing import image
predict_image('/kaggle/input/intel-image-classification/seg_pred/seg_pred/10092.jpg', model)
```

```
1/1 [==============================] - 0s 27ms/step
```




```
Start coding or generate with AI.
```