Multiclass Image Classification Using CNN

In this notebook I have shown how a simple CNN is implemented on a multiclass image classification problem. I have covered

1. How to create a CNN Model and Train it.

2. How to evaluate the model on test set using different classification metrics.

3. How to visualize the images present in the training and test set.

I hope you find this kernel helpful and some **UPVOTES** would be very much appreciated

1. Import the Required Libraries

```
from google.colab import drive

drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import cv2
import os
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
import warnings
warnings.filterwarnings('ignore')
```

```
from sklearn.metrics import confusion_matrix, classification_report

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, BatchNormalization, Conv2D, Dense, Dropout, Flatten, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
```

2. Load the Image Training and Validation Datasets

i. Get the Image Dataset Paths

```
train_dataset_path = '/content/drive/MyDrive/seg_train'
validation_dataset_path = '/content/drive/MyDrive/seg_test'
```

ii. Load Image Datasets and Apply Augmentations

Since the images present in the datasets are 150x150px in size, the image height and width are taken as 150, 150 respectively. The batch size value can be changed if required.

```
IMG_WIDTH = 150
IMG_HEIGHT = 150
BATCH_SIZE = 32
```

Loading the training dataset and applying augmentations on it.

```
train_datagen = ImageDataGenerator(rescale=1.0/255,
                                  zoom_range=0.2,
                                  width_shift_range=0.2,
                                  height_shift_range=0.2,
                                  fill_mode='nearest')
train_generator = train_datagen.flow_from_directory(train_dataset_path,
                                                   target_size=(IMG_WIDTH, IMG_HEIGHT),
                                                   batch_size=BATCH_SIZE,
                                                   class_mode='categorical',
                                                   shuffle=True)
```

```
Found 120 images belonging to 1 classes.
```

Loading the validation dataset.

```
validation_datagen = ImageDataGenerator(rescale=1.0/255)
validation_generator = validation_datagen.flow_from_directory(validation_dataset_path,
                                                             target_size=(IMG_WIDTH, IMG_HEIGHT),
                                                             batch_size=BATCH_SIZE,
                                                             class_mode='categorical',
                                                             shuffle=True)
```

```
Found 120 images belonging to 1 classes.
```

iii. Get the Label Mappings

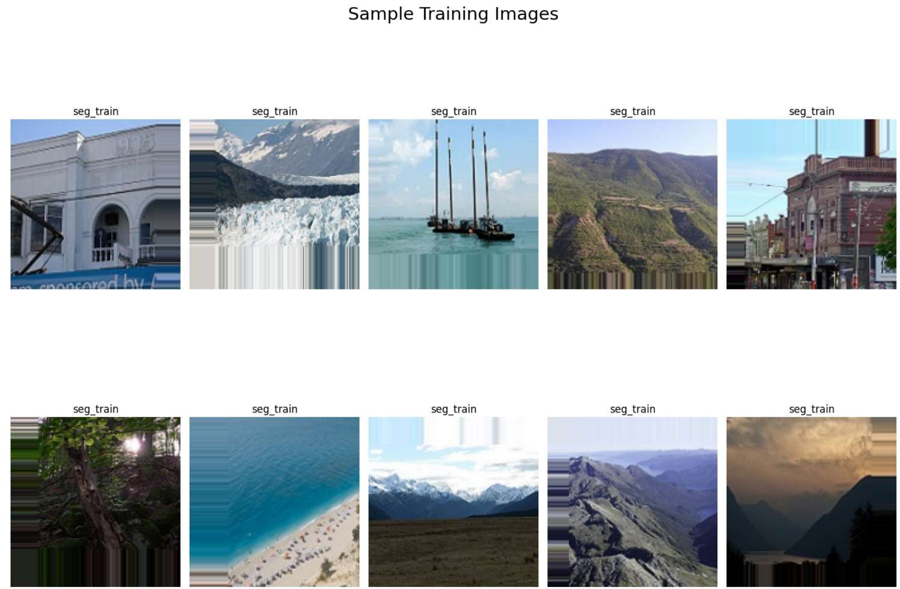
The labels dictionary is made in order to retrive the class names against the label indices used for training the model

```
labels = {value: key for key, value in train_generator.class_indices.items()}

print("Label Mappings for classes present in the training and validation datasets\n")
for key, value in labels.items():
    print(f"{key} : {value}")
```

```
Label Mappings for classes present in the training and validation datasets

0 : seg_train
```

3. Plotting Sample Training Images

```
fig, ax = plt.subplots(nrows=2, ncols=5, figsize=(15, 12))
idx = 0

for i in range(2):
    for j in range(5):
        label = labels[np.argmax(train_generator[0][1][idx])]
        ax[i, j].set_title(f"{label}")
        ax[i, j].imshow(train_generator[0][0][idx][:, :, :])
        ax[i, j].axis("off")
        idx += 1

plt.tight_layout()
plt.suptitle("Sample Training Images", fontsize=21)
plt.show()
```


Sample Training Images
seg_train
seg_train
seg_train
seg_train
seg_train
seg_train
seg_train
seg_train
seg_train
seg_train

4. Training a CNN Model

Since the training dataset is ready let's create a simple CNN Model to train on the image datasets

i. Create a CNN Model

```
def create_model():
    model = Sequential([
        Conv2D(filters=128, kernel_size=(5, 5), padding='valid', input_shape=(IMG_WIDTH, IMG_HEIGHT, 3)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),

        Conv2D(filters=64, kernel_size=(3, 3), padding='valid', kernel_regularizer=l2(0.00005)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),

        Conv2D(filters=32, kernel_size=(3, 3), padding='valid', kernel_regularizer=l2(0.00005)),
        Activation('relu'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),

        Flatten(),

        Dense(units=256, activation='relu'),
        Dropout(0.5),
        Dense(units=6, activation='softmax')
    ])

    return model


cnn_model = create_model()


print(cnn_model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
conv2d_3 (Conv2D)           (None, 146, 146, 128)     9728

activation_3 (Activation)   (None, 146, 146, 128)     0

max_pooling2d_3 (MaxPoolin  (None, 73, 73, 128)       0
g2D)

batch_normalization_3 (Bat  (None, 73, 73, 128)       512
chNormalization)

conv2d_4 (Conv2D)           (None, 71, 71, 64)        73792

activation_4 (Activation)   (None, 71, 71, 64)        0

max_pooling2d_4 (MaxPoolin  (None, 35, 35, 64)        0
g2D)

batch_normalization_4 (Bat  (None, 35, 35, 64)        256
chNormalization)

conv2d_5 (Conv2D)           (None, 33, 33, 32)        18464

activation_5 (Activation)   (None, 33, 33, 32)        0

max_pooling2d_5 (MaxPoolin  (None, 16, 16, 32)        0
g2D)

batch_normalization_5 (Bat  (None, 16, 16, 32)        128
chNormalization)

flatten_1 (Flatten)         (None, 8192)              0

dense_2 (Dense)             (None, 256)               2097408

dropout_1 (Dropout)         (None, 256)               0

dense_3 (Dense)             (None, 6)                 1542

=================================================================
```

```
Total params: 2201830 (8.40 MB)
Trainable params: 2201382 (8.40 MB)
Non-trainable params: 448 (1.75 KB)
_________________________________________________________________
None
```

ii. Defining Callbacks

A callback is an object that can perform actions at various stages of training (e.g. at the start or end of an epoch, before or after a single batch, etc)

a. Reduce Learning Rate on Plateau

Is used to reduce the learning rate when a metric has stopped improving.

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=np.sqrt(0.1), patience=5)
```

iii. Defining the Optimizer

```
optimizer = Adam(learning_rate=0.001)
```

iv. Compile the Model

```
cnn_model.compile(optimizer=optimizer, loss=CategoricalCrossentropy(), metrics=['accuracy'])
```

v. Training the Model

```
history = cnn_model.fit(train_generator, epochs=5, validation_data=validation_generator,
                        verbose=2,
                        callbacks=[reduce_lr])
```

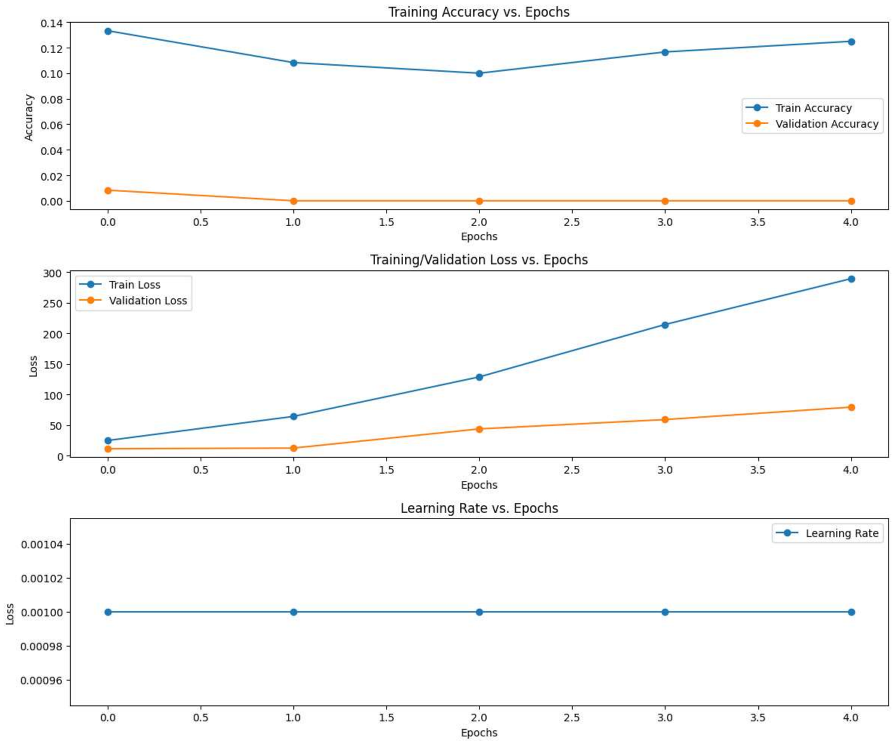
```
Epoch 1/5
4/4 - 31s - loss: 24.9690 - accuracy: 0.1333 - val_loss: 11.5046 - val_accuracy: 0.0083 - lr: 0.0010 - 31s/epoch - 8s/step
Epoch 2/5
4/4 - 20s - loss: 64.3784 - accuracy: 0.1083 - val_loss: 12.6371 - val_accuracy: 0.0000e+00 - lr: 0.0010 - 20s/epoch - 5s/step
Epoch 3/5
4/4 - 18s - loss: 128.9673 - accuracy: 0.1000 - val_loss: 43.9791 - val_accuracy: 0.0000e+00 - lr: 0.0010 - 18s/epoch - 5s/step
Epoch 4/5
4/4 - 20s - loss: 214.6829 - accuracy: 0.1167 - val_loss: 59.2378 - val_accuracy: 0.0000e+00 - lr: 0.0010 - 20s/epoch - 5s/step
Epoch 5/5
4/4 - 19s - loss: 289.5929 - accuracy: 0.1250 - val_loss: 79.5754 - val_accuracy: 0.0000e+00 - lr: 0.0010 - 19s/epoch - 5s/step
```

5. Plotting the Model Metrics

i. Plotting training and validation accuracy, loss and learning rate

```
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

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

learning_rate = history.history['lr']


fig, ax = plt.subplots(nrows=3, ncols=1, figsize=(12, 10))

ax[0].set_title('Training Accuracy vs. Epochs')
ax[0].plot(train_accuracy, 'o-', label='Train Accuracy')
```

```
ax[0].plot(val_accuracy, 'o-', label='Validation Accuracy')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Accuracy')
ax[0].legend(loc='best')

ax[1].set_title('Training/Validation Loss vs. Epochs')
ax[1].plot(train_loss, 'o-', label='Train Loss')
ax[1].plot(val_loss, 'o-', label='Validation Loss')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].legend(loc='best')

ax[2].set_title('Learning Rate vs. Epochs')
ax[2].plot(learning_rate, 'o-', label='Learning Rate')
ax[2].set_xlabel('Epochs')
ax[2].set_ylabel('Loss')
ax[2].legend(loc='best')

plt.tight_layout()
plt.show()
```


Training Accuracy vs. Epochs
Accuracy
Epochs
Train Accuracy
Validation Accuracy
Training/Validation Loss vs. Epochs
Loss
Epochs
Train Loss
Validation Loss
Learning Rate vs. Epochs
Loss
Epochs
Learning Rate

6. Testing the Model on Test Set

Testing the model on the validation dataset because a seperate dataset for testing is not available.

```
test_dataset = '/content/drive/MyDrive/seg_pred'
```

```
test_datagen = ImageDataGenerator(rescale=1.0/255)

test_generator = test_datagen.flow_from_directory(test_dataset,
                                                  shuffle=False,
                                                  batch_size=BATCH_SIZE,
                                                  target_size = (IMG_WIDTH, IMG_HEIGHT),
                                                  class_mode='categorical')
```

```
Found 20 images belonging to 1 classes.
```

7. Model Prediction on the Test Dataset

```
predictions = cnn_model.predict(test_generator)
```

```
1/1 [==============================] - 1s 750ms/step
```

```
test_loss, test_accuracy = cnn_model.evaluate(test_generator, batch_size=BATCH_SIZE)
```

```
1/1 [==============================] - 1s 647ms/step - loss: 75.8547 - accuracy: 0.0000e+00
```

```
print(f"Test Loss:     {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

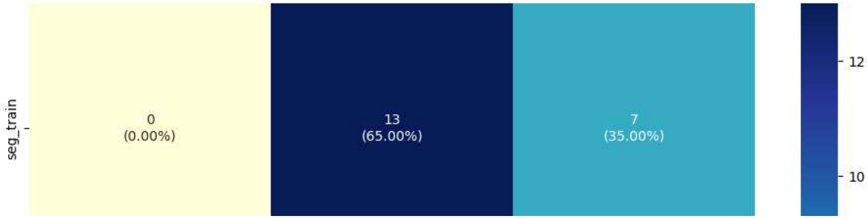
```
Test Loss:     75.85465240478516
Test Accuracy: 0.0
```

The test loss and test accuracy is the same as validation loss and validation accuracy at the last step since the testing and validation datasets are same.

8. Plotting the Classification Metrics

i. Confusion Matrix

```
y_pred = np.argmax(predictions, axis=1)
y_true = test_generator.classes


cf_mtx = confusion_matrix(y_true, y_pred)

group_counts = ["{0:0.0f}".format(value) for value in cf_mtx.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in cf_mtx.flatten()/np.sum(cf_mtx)]
box_labels = [f"{v1}\n({v2})" for v1, v2 in zip(group_counts, group_percentages)]
box_labels = np.asarray(box_labels).reshape(3, 3)

plt.figure(figsize = (12, 10))
sns.heatmap(cf_mtx, xticklabels=labels.values(), yticklabels=labels.values(),
           cmap="YlGnBu", fmt="", annot=box_labels)
plt.xlabel('Predicted Classes')
plt.ylabel('True Classes')
plt.show()
```




```
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      20.0
           1       0.00      0.00      0.00       0.0
           4       0.00      0.00      0.00       0.0

    accuracy                           0.00      20.0
   macro avg       0.00      0.00      0.00      20.0
weighted avg       0.00      0.00      0.00      20.0
```

9. Wrong Predictions

Let's see where the model has given wrong predictions and what were the actual predictions on those images.

```
errors = (y_true - y_pred != 0)
y_true_errors = y_true[errors]
y_pred_errors = y_pred[errors]
```

```
test_images = test_generator.filenames
```