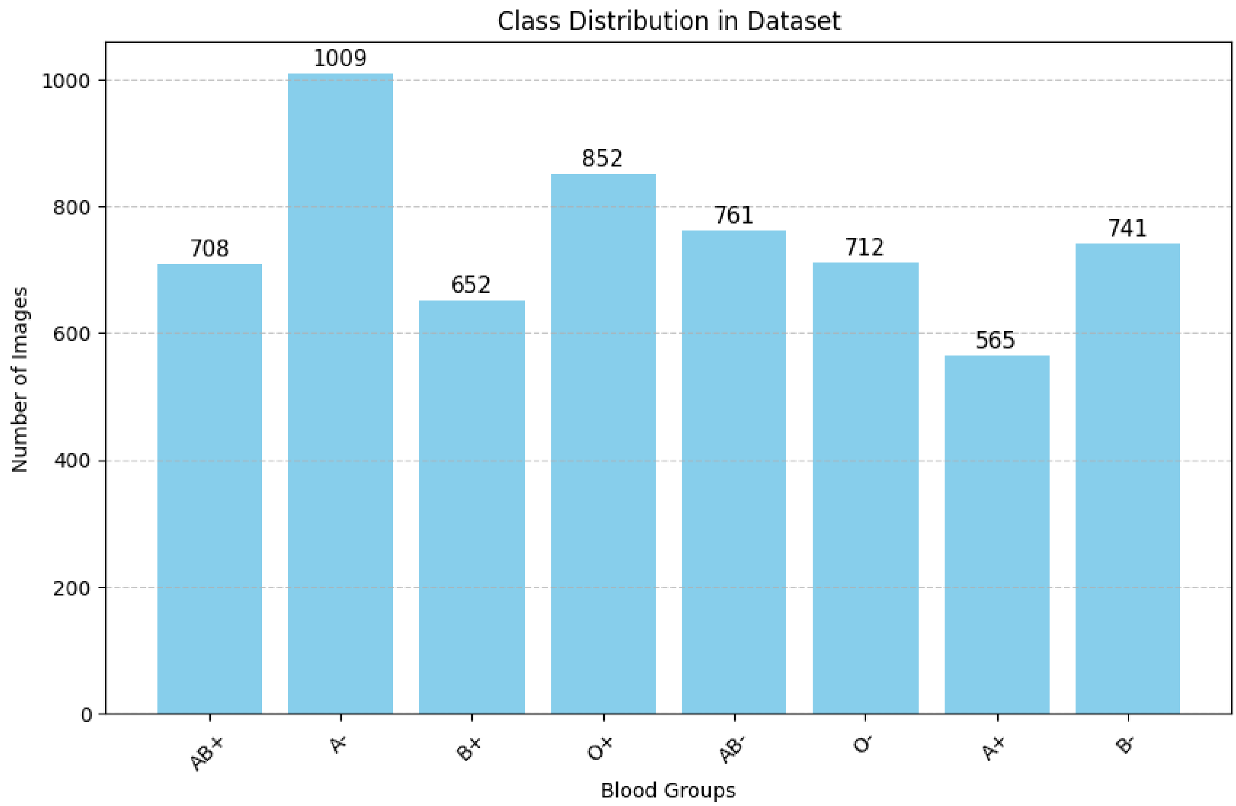
```python
import os
os.environ["TF_CPP_MIN_LOG_LEVEL"] = '2'

import tensorflow as tf
import numpy as np
import seaborn as sns
from collections import Counter
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import shutil
from sklearn.utils import resample
from tensorflow.keras.preprocessing.image import
load_img,img_to_array,save_img
from tensorflow.keras.utils import image_dataset_from_directory
from sklearn.metrics import classification_report,confusion_matrix
import matplotlib.pyplot as plt

dataset_path = "/home/rgukt/Downloads/project/dataset_blood_group"

BATCH_SIZE = 32

dataset = image_dataset_from_directory(
    dataset_path,
    labels = "inferred",
    label_mode = "int",
    image_size = (64,64),
    batch_size = BATCH_SIZE,
    shuffle = True
)
```

```
Found 6000 files belonging to 8 classes.
```

```python
# Chcek class distribution
class_names = dataset.class_names
class_counts = Counter()
for _, labels in dataset.unbatch():
    class_counts[int(labels.numpy())] += 1

print("Class Distribution")
for i,count in class_counts.items():
    print(f"{class_names[i]} : {count}")
```

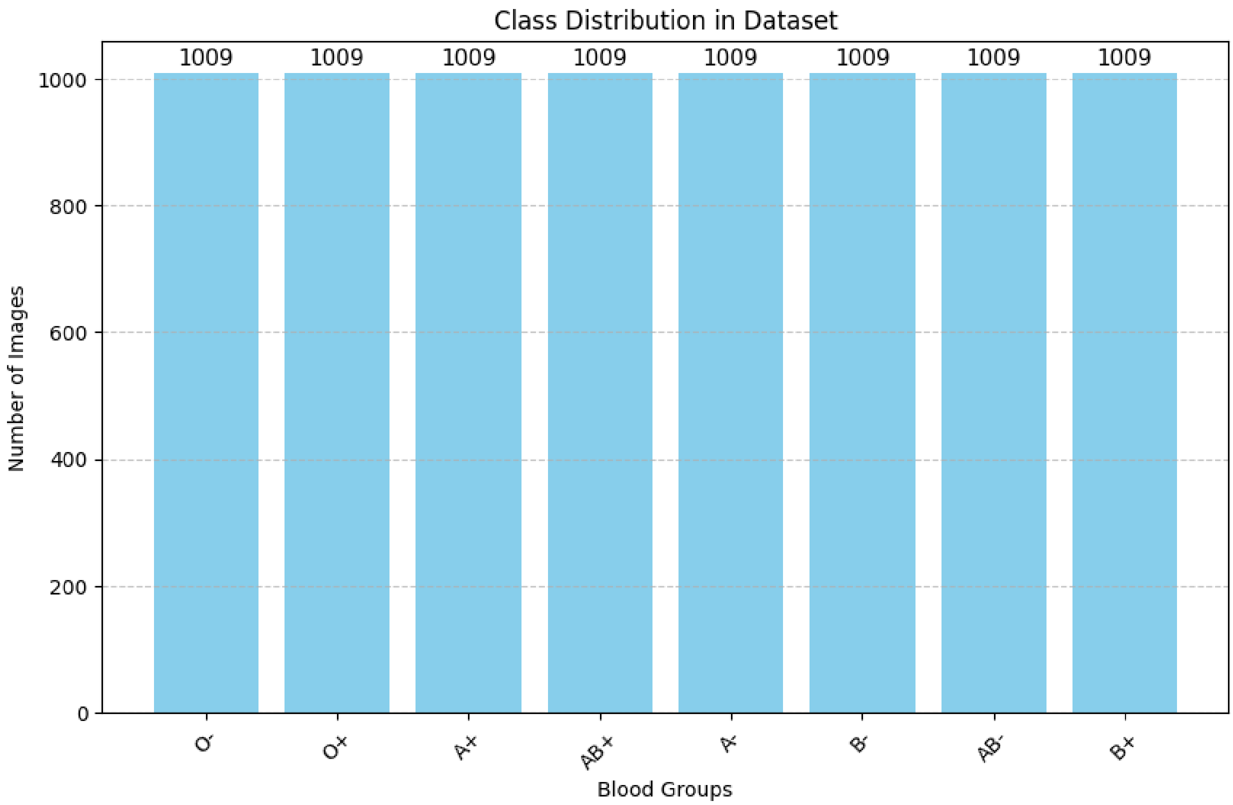
```
Class Distribution
AB+ : 708
A- : 1009
B+ : 652
O+ : 852
AB- : 761
O- : 712
A+ : 565
B- : 741
```

```
def plot_class_distribution(class_names,class_counts):
   # class_names (list): List of class names.
   # class_counts (dict): Dictionary where keys are class indices and
values are counts

    classes = [class_names[i] for i in class_counts.keys()]
    counts = [class_counts[i] for i in class_counts.keys()]

    plt.figure(figsize=(10,6))
    bars = plt.bar(classes, counts, color='skyblue')
    plt.xlabel("Blood Groups")
    plt.ylabel('Number of Images')
    plt.title('Class Distribution in Dataset')
    plt.xticks(rotation=45)  # Rotate class names for better
readability
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    for bar in bars:
        height = bar.get_height()
        plt.text(
            bar.get_x() + bar.get_width() / 2,  # X position (center
of the bar)
            height + 5,  # Y position (slightly above the bar)
            f'{int(height)}',  # Text to display (the count value)
            ha='center',  # Horizontal alignment
            va='bottom',  # Vertical alignment
            fontsize=11
        )
    plt.show()

plot_class_distribution(class_names,class_counts)
```




```
max_count = max(class_counts.values())

def oversample_class(class_id,count,max_count):
    # dataset is unbatched for filtering
    unbatched_ds = dataset.unbatch()
    class_ds = unbatched_ds.filter(lambda
img,lbl:tf.equal(lbl,class_id))

    # filter dataset for specific class
    repeat_factor = max_count // count + (max_count % count > 0)

    # repeat the dataset to match desired count
    return class_ds.repeat(repeat_factor).take(max_count)

# balance the dataset
balanced_ds = []
for class_id,count in class_counts.items():
    balanced_ds.append(oversample_class(class_id,count,max_count))

# combine balanced datasets
balanced_dataset = tf.data.Dataset.sample_from_datasets(balanced_ds)

# check balanced class distribution
balanced_class_counts = Counter([int(lbl.numpy()) for _, lbl in
balanced_dataset])
```

```python
plot_class_distribution(class_names,balanced_class_counts)

# batch the balanced dataset
balanced_dataset =
balanced_dataset.batch(BATCH_SIZE,drop_remainder=True)
```




```python
for sample in balanced_dataset.take(10):
    print(sample[0].shape)
```

```
(32, 64, 64, 3)
(32, 64, 64, 3)
(32, 64, 64, 3)
(32, 64, 64, 3)
(32, 64, 64, 3)
(32, 64, 64, 3)
(32, 64, 64, 3)
(32, 64, 64, 3)
(32, 64, 64, 3)
(32, 64, 64, 3)
```

```python
balanced_ds_unbatched = balanced_dataset.unbatch()
dataset_size = sum(1 for _ in balanced_ds_unbatched)
print(f"Total dataset size:{dataset_size}")
```

```
Total dataset size:8064
```

```python
# unbatch to work at sample level
balanced_ds_unbatched = balanced_dataset.unbatch()

# Desired ratios
train_ratio = 5632 / dataset_size  #  0.701
val_ratio = 1600 / dataset_size    #  0.199

# compute sizes based on dataset size and desired splits
train_size = int(train_ratio * dataset_size)
val_size = int(val_ratio * dataset_size)

# split dataset into training,test and validation
train_ds = balanced_ds_unbatched.take(train_size)
val_test_ds = balanced_ds_unbatched.skip(train_size)
val_ds = val_test_ds.take(val_size)
test_ds = val_test_ds.skip(val_size)

# rebatch the datasets after splitting
train_ds = train_ds.batch(BATCH_SIZE,drop_remainder=True)
val_ds = val_ds.batch(BATCH_SIZE,drop_remainder=True)
test_ds = test_ds.batch(BATCH_SIZE,drop_remainder=True)

# check the no.of batches in each dataset
train_batch_count =sum(1 for _ in train_ds)
val_batch_count =sum(1 for _ in val_ds)
test_batch_count =sum(1 for _ in test_ds)

print(f"Training dataset size: {train_batch_count * BATCH_SIZE}")
print(f"Validation dataset size: {val_batch_count * BATCH_SIZE}")
print(f"Testing dataset size: {test_batch_count * BATCH_SIZE}")
```

```
Training dataset size: 5632
Validation dataset size: 1600
Testing dataset size: 832
```

```python
def create_high_accuracy_model():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(32,
(3,3),activation="relu",padding="same",input_shape=(64,64,3)),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Dropout(0.3),

        tf.keras.layers.Conv2D(64,
(3,3),activation="relu",padding="same"),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Dropout(0.4),

        tf.keras.layers.Conv2D(128,
(3,3),activation="relu",padding="same"),
        tf.keras.layers.MaxPooling2D(2,2),
```

```
        tf.keras.layers.Dropout(0.4),

        tf.keras.layers.Conv2D(256,
(3,3),activation="relu",padding="same"),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Dropout(0.4),

        tf.keras.layers.Conv2D(512,
(3,3),activation="relu",padding="same"),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Dropout(0.5),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(1024,activation="relu"),
        tf.keras.layers.Dropout(0.5),

        tf.keras.layers.Dense(len(class_names),activation="softmax")
    ])

    model.compile(optimizer="adam",
                  loss="sparse_categorical_crossentropy",
                  metrics=["accuracy"])

    return model

high_accuracy_model = create_high_accuracy_model()
```

```
/home/rgukt/.local/lib/python3.10/site-packages/keras/src/layers/
convolutional/base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

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

# ReduceLROnPlateau callback to reduce learning rate when validation
loss pateau
reduce_lr = ReduceLROnPlateau(
    monitor = "val_loss",
    factor = 0.5,
    patience = 3,
    verbose = 1,
    min_lr = 1e-6
)

# Early stopping callback to stop training when validation loss
doesn't improve
early_stop = EarlyStopping(
```

```
    monitor = "val_loss",
    patience = 5,
    verbose = 1,
    restore_best_weights = True
)

# train the model

history_high_acc = high_accuracy_model.fit(
    train_ds,
    validation_data = val_ds,
    epochs = 50,
    callbacks = [reduce_lr,early_stop]
)
```

```
Epoch 1/50
    176/Unknown 100s 544ms/step - accuracy: 0.1313 - loss: 29.5351
```

```
/home/rgukt/.local/lib/python3.10/site-packages/keras/src/trainers/
epoch_iterator.py:151: UserWarning: Your input ran out of data;
interrupting training. Make sure that your dataset or generator can
generate at least `steps_per_epoch * epochs` batches. You may need to
use the `.repeat()` function when building your dataset.
  self._interrupted_warning()
```

```
176/176 ━━━━━━━━━━━━━━━━━━━━ 110s 600ms/step - accuracy: 0.1313 -
loss: 29.4139 - val_accuracy: 0.1379 - val_loss: 2.0383 -
learning_rate: 0.0010
Epoch 2/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 98s 557ms/step - accuracy: 0.1239 - loss:
2.0853 - val_accuracy: 0.1379 - val_loss: 2.0388 - learning_rate:
0.0010
Epoch 3/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 108s 616ms/step - accuracy: 0.1378 -
loss: 2.0809 - val_accuracy: 0.1127 - val_loss: 2.0383 -
learning_rate: 0.0010
Epoch 4/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 109s 615ms/step - accuracy: 0.1410 -
loss: 2.0762 - val_accuracy: 0.2702 - val_loss: 1.9404 -
learning_rate: 0.0010
Epoch 5/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 143s 815ms/step - accuracy: 0.2218 -
loss: 1.9811 - val_accuracy: 0.3241 - val_loss: 1.8127 -
learning_rate: 0.0010
Epoch 6/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 96s 546ms/step - accuracy: 0.3495 - loss:
1.7295 - val_accuracy: 0.3860 - val_loss: 1.6995 - learning_rate:
0.0010
Epoch 7/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 96s 544ms/step - accuracy: 0.4276 - loss:
```

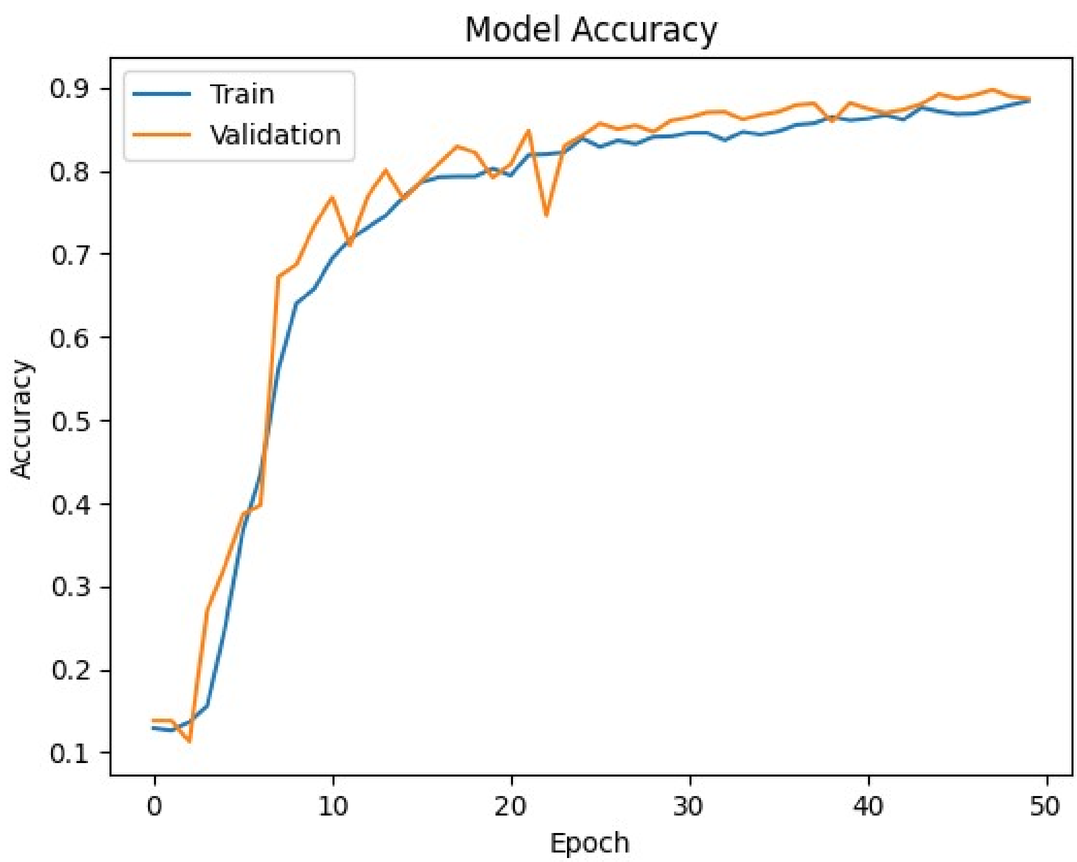
```
1.5026 - val_accuracy: 0.3977 - val_loss: 1.6098 - learning_rate:
0.0010
Epoch 8/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 145s 825ms/step - accuracy: 0.5476 -
loss: 1.2078 - val_accuracy: 0.6722 - val_loss: 1.3261 -
learning_rate: 0.0010
Epoch 9/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 100s 565ms/step - accuracy: 0.6358 -
loss: 0.9817 - val_accuracy: 0.6869 - val_loss: 1.1502 -
learning_rate: 0.0010
Epoch 10/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 99s 559ms/step - accuracy: 0.6532 - loss:
0.9141 - val_accuracy: 0.7335 - val_loss: 0.9256 - learning_rate:
0.0010
Epoch 11/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 98s 553ms/step - accuracy: 0.6953 - loss:
0.8198 - val_accuracy: 0.7678 - val_loss: 0.7568 - learning_rate:
0.0010
Epoch 12/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 97s 548ms/step - accuracy: 0.7204 - loss:
0.7641 - val_accuracy: 0.7102 - val_loss: 0.9307 - learning_rate:
0.0010
Epoch 13/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 96s 545ms/step - accuracy: 0.7270 - loss:
0.7432 - val_accuracy: 0.7690 - val_loss: 0.8217 - learning_rate:
0.0010
Epoch 14/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 484ms/step - accuracy: 0.7453 - loss:
0.6977
Epoch 14: ReduceLROnPlateau reducing learning rate to
0.0005000000237487257.
176/176 ━━━━━━━━━━━━━━━━━━━━ 94s 535ms/step - accuracy: 0.7453 - loss:
0.6976 - val_accuracy: 0.8002 - val_loss: 0.8116 - learning_rate:
0.0010
Epoch 15/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 132s 478ms/step - accuracy: 0.7732 -
loss: 0.6042 - val_accuracy: 0.7653 - val_loss: 0.7927 -
learning_rate: 5.0000e-04
Epoch 16/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 85s 480ms/step - accuracy: 0.7955 - loss:
0.5797 - val_accuracy: 0.7874 - val_loss: 0.6870 - learning_rate:
5.0000e-04
Epoch 17/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 85s 483ms/step - accuracy: 0.7896 - loss:
0.5590 - val_accuracy: 0.8088 - val_loss: 0.6786 - learning_rate:
5.0000e-04
Epoch 18/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 85s 483ms/step - accuracy: 0.8012 - loss:
0.5242 - val_accuracy: 0.8290 - val_loss: 0.5978 - learning_rate:
```

```
5.0000e-04
Epoch 19/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 142s 808ms/step - accuracy: 0.7990 -
loss: 0.5228 - val_accuracy: 0.8217 - val_loss: 0.6483 -
learning_rate: 5.0000e-04
Epoch 20/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 85s 484ms/step - accuracy: 0.8107 - loss:
0.5063 - val_accuracy: 0.7917 - val_loss: 0.6717 - learning_rate:
5.0000e-04
Epoch 21/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 436ms/step - accuracy: 0.7929 - loss:
0.5195
Epoch 21: ReduceLROnPlateau reducing learning rate to
0.0002500000118743628.
176/176 ━━━━━━━━━━━━━━━━━━━━ 85s 485ms/step - accuracy: 0.7929 - loss:
0.5195 - val_accuracy: 0.8076 - val_loss: 0.6682 - learning_rate:
5.0000e-04
Epoch 22/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 86s 486ms/step - accuracy: 0.8183 - loss:
0.4763 - val_accuracy: 0.8480 - val_loss: 0.5052 - learning_rate:
2.5000e-04
Epoch 23/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 86s 487ms/step - accuracy: 0.8257 - loss:
0.4477 - val_accuracy: 0.7457 - val_loss: 0.7276 - learning_rate:
2.5000e-04
Epoch 24/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 86s 487ms/step - accuracy: 0.8226 - loss:
0.4666 - val_accuracy: 0.8297 - val_loss: 0.5370 - learning_rate:
2.5000e-04
Epoch 25/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 86s 487ms/step - accuracy: 0.8413 - loss:
0.4172 - val_accuracy: 0.8419 - val_loss: 0.5022 - learning_rate:
2.5000e-04
Epoch 26/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 142s 809ms/step - accuracy: 0.8305 -
loss: 0.4339 - val_accuracy: 0.8566 - val_loss: 0.4806 -
learning_rate: 2.5000e-04
Epoch 27/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 86s 489ms/step - accuracy: 0.8460 - loss:
0.4017 - val_accuracy: 0.8499 - val_loss: 0.4400 - learning_rate:
2.5000e-04
Epoch 28/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 142s 808ms/step - accuracy: 0.8359 -
loss: 0.4268 - val_accuracy: 0.8542 - val_loss: 0.4653 -
learning_rate: 2.5000e-04
Epoch 29/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 86s 489ms/step - accuracy: 0.8544 - loss:
0.3814 - val_accuracy: 0.8468 - val_loss: 0.4746 - learning_rate:
2.5000e-04
```

```
Epoch 30/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 86s 489ms/step - accuracy: 0.8429 - loss:
0.4177 - val_accuracy: 0.8603 - val_loss: 0.4177 - learning_rate:
2.5000e-04
Epoch 31/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 142s 808ms/step - accuracy: 0.8514 -
loss: 0.3860 - val_accuracy: 0.8640 - val_loss: 0.4047 -
learning_rate: 2.5000e-04
Epoch 32/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 86s 489ms/step - accuracy: 0.8487 - loss:
0.3926 - val_accuracy: 0.8701 - val_loss: 0.3967 - learning_rate:
2.5000e-04
Epoch 33/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 86s 489ms/step - accuracy: 0.8498 - loss:
0.4017 - val_accuracy: 0.8707 - val_loss: 0.4444 - learning_rate:
2.5000e-04
Epoch 34/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 86s 490ms/step - accuracy: 0.8477 - loss:
0.3943 - val_accuracy: 0.8615 - val_loss: 0.4305 - learning_rate:
2.5000e-04
Epoch 35/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 442ms/step - accuracy: 0.8508 - loss:
0.3873
Epoch 35: ReduceLROnPlateau reducing learning rate to
0.0001250000059371814.
176/176 ━━━━━━━━━━━━━━━━━━━━ 87s 491ms/step - accuracy: 0.8508 - loss:
0.3873 - val_accuracy: 0.8670 - val_loss: 0.4480 - learning_rate:
2.5000e-04
Epoch 36/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 87s 493ms/step - accuracy: 0.8466 - loss:
0.3821 - val_accuracy: 0.8707 - val_loss: 0.4107 - learning_rate:
1.2500e-04
Epoch 37/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 86s 489ms/step - accuracy: 0.8599 - loss:
0.3593 - val_accuracy: 0.8787 - val_loss: 0.3863 - learning_rate:
1.2500e-04
Epoch 38/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 142s 808ms/step - accuracy: 0.8583 -
loss: 0.3551 - val_accuracy: 0.8805 - val_loss: 0.3999 -
learning_rate: 1.2500e-04
Epoch 39/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 88s 502ms/step - accuracy: 0.8641 - loss:
0.3467 - val_accuracy: 0.8591 - val_loss: 0.3939 - learning_rate:
1.2500e-04
Epoch 40/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 87s 491ms/step - accuracy: 0.8647 - loss:
0.3457 - val_accuracy: 0.8811 - val_loss: 0.3720 - learning_rate:
1.2500e-04
Epoch 41/50
```

```
176/176 ━━━━━━━━━━━━━━━━━━━━ 87s 494ms/step - accuracy: 0.8709 - loss:
0.3207 - val_accuracy: 0.8744 - val_loss: 0.3807 - learning_rate:
1.2500e-04
Epoch 42/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 87s 494ms/step - accuracy: 0.8761 - loss:
0.3350 - val_accuracy: 0.8695 - val_loss: 0.4092 - learning_rate:
1.2500e-04
Epoch 43/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 442ms/step - accuracy: 0.8690 - loss:
0.3207
Epoch 43: ReduceLROnPlateau reducing learning rate to
6.25000029685907e-05.
176/176 ━━━━━━━━━━━━━━━━━━━━ 87s 491ms/step - accuracy: 0.8689 - loss:
0.3208 - val_accuracy: 0.8732 - val_loss: 0.3742 - learning_rate:
1.2500e-04
Epoch 44/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 87s 493ms/step - accuracy: 0.8761 - loss:
0.3197 - val_accuracy: 0.8799 - val_loss: 0.3574 - learning_rate:
6.2500e-05
Epoch 45/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 87s 495ms/step - accuracy: 0.8728 - loss:
0.3314 - val_accuracy: 0.8922 - val_loss: 0.3501 - learning_rate:
6.2500e-05
Epoch 46/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 87s 496ms/step - accuracy: 0.8757 - loss:
0.3241 - val_accuracy: 0.8866 - val_loss: 0.3630 - learning_rate:
6.2500e-05
Epoch 47/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 87s 493ms/step - accuracy: 0.8744 - loss:
0.3103 - val_accuracy: 0.8909 - val_loss: 0.3452 - learning_rate:
6.2500e-05
Epoch 48/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 88s 497ms/step - accuracy: 0.8789 - loss:
0.3028 - val_accuracy: 0.8971 - val_loss: 0.3401 - learning_rate:
6.2500e-05
Epoch 49/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 87s 494ms/step - accuracy: 0.8835 - loss:
0.3034 - val_accuracy: 0.8891 - val_loss: 0.3413 - learning_rate:
6.2500e-05
Epoch 50/50
176/176 ━━━━━━━━━━━━━━━━━━━━ 87s 492ms/step - accuracy: 0.8880 - loss:
0.2889 - val_accuracy: 0.8866 - val_loss: 0.3342 - learning_rate:
6.2500e-05
Restoring model weights from the end of the best epoch: 50.

high_accuracy_eval = high_accuracy_model.evaluate(val_ds)
print(f"High Accuracy Model - Loss: {high_accuracy_eval[0]},Accuracy:
{high_accuracy_eval[1]}")
```

```
50/50 ━━━━━━━━━━━━━━━━━━━━ 9s 114ms/step - accuracy: 0.9009 - loss: 0.3369
High Accuracy Model - Loss: 0.3332694470882416,Accuracy: 0.8860294222831726

def plot_accuracy(history):
    plt.plot(history.history["accuracy"])
    plt.plot(history.history["val_accuracy"])
    plt.title("Model Accuracy")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.legend(["Train","Validation"],loc="upper left")
    plt.show()

plot_accuracy(history_high_acc)
```

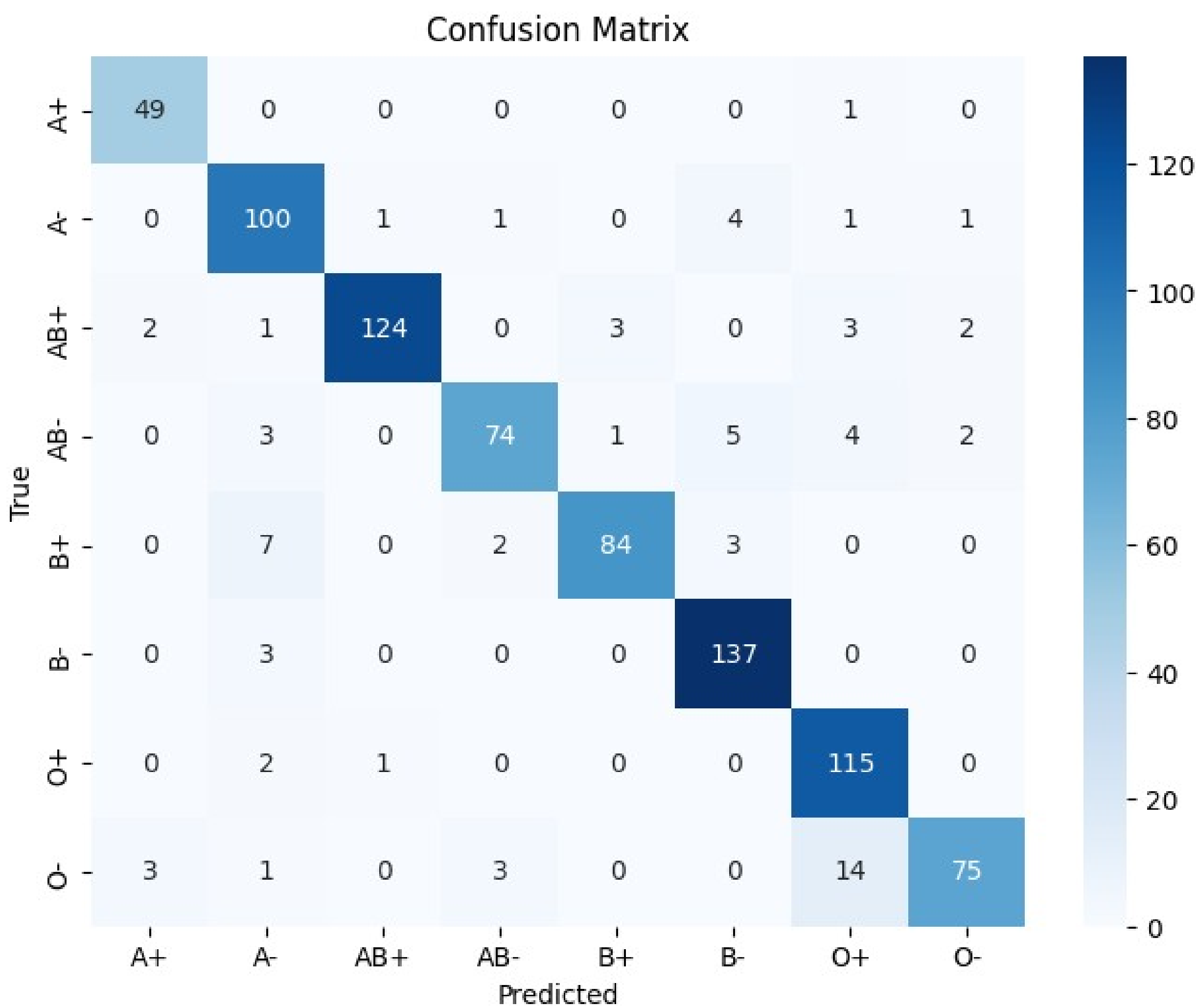



```
y_true = []
y_pred = []

for images, labels in test_ds:
    predictions = high_accuracy_model.predict(images)
    prediction_labels = np.argmax(predictions,axis=1)
    y_true.extend(labels.numpy())
```

```
    y_pred.extend(prediction_labels)

y_true = np.array(y_true)
y_pred = np.array(y_pred)

report = classification_report(y_true,y_pred,target_names =
class_names)
print("Classification Report:")
print(report)

# confusion matrix
conf_matrix =confusion_matrix(y_true,y_pred)

# plot confusion matrix
plt.figure(figsize =(8,6))
sns.heatmap(conf_matrix,annot=True,fmt="d",cmap="Blues",xticklabels=cl
ass_names,yticklabels=class_names)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 145ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 139ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 137ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 130ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 128ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 129ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 126ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 129ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 125ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 128ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 144ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 144ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 238ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 148ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 139ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 131ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 144ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 141ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 143ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
Classification Report:
              precision    recall  f1-score   support
```

A+	0.91	0.98	0.94	50
A-	0.85	0.93	0.89	108
AB+	0.98	0.92	0.95	135
AB-	0.93	0.83	0.88	89
B+	0.95	0.88	0.91	96
B-	0.92	0.98	0.95	140
O+	0.83	0.97	0.90	118
O-	0.94	0.78	0.85	96
accuracy			0.91	832
macro avg	0.91	0.91	0.91	832
weighted avg	0.92	0.91	0.91	832



```
high_accuracy_model.save("home/rgukt/Downloads/project/model/
model.h5")
print("Model saved as HDFS format.")
```

```
WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.

Model saved as HDFS format.
```