Start coding or generate with AI.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.optimizers import Adam



# ImageDataGenerator for loading and augmenting images
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)


# Directories for dataset
train_dir = '/content/drive/MyDrive/Classroom/split_minip/train'
val_dir = '/content/drive/MyDrive/Classroom/split_minip/val'
test_dir = '/content/drive/MyDrive/Classroom/split_minip/test'


# Image generators for loading images with resizing to 224x224
train_gen = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)
val_gen = val_datagen.flow_from_directory(
    val_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)
test_gen = test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 5600 images belonging to 8 classes.
Found 1201 images belonging to 8 classes.
Found 1162 images belonging to 8 classes.
```

```
# Load ResNet50 with pre-trained weights, excluding the top layers
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_no
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

```
# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False


# Add custom layers on top of the base model
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(8, activation='softmax')(x)



# Create the model
model = Model(inputs=base_model.input, outputs=predictions)


# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

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


# Define paths to save models
checkpoint_path = '/content/drive/MyDrive/blood_group_best_model.keras'  # Best model
final_model_path = '/content/drive/MyDrive/blood_group_final_model.keras'  # Final model after training


# Define callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)


model_checkpoint = ModelCheckpoint(
    checkpoint_path,
    monitor='val_loss',
    save_best_only=True,
    verbose=1  # Show a message when saving the best model
)


# Train the model
history = model.fit(
    train_gen,  # Use the train generator
    validation_data=val_gen,  # Use the validation generator
    epochs=30,  # You can adjust the number of epochs
    callbacks=[early_stopping, model_checkpoint]
)
```

```
Epoch 30: val_loss improved from 1.07372 to 1.07358, saving model to /content/drive/MyDrive/blood_group_best_model.keras
175/175 ━━━━━━━━━━━━━━━━━━━━ 40s 164ms/step - accuracy: 0.6369 - loss: 1.0629 - val_accuracy: 0.6270 - val_loss: 1.0736
```

```
# Save the final model explicitly after training (optional but recommended)
model.save(final_model_path)
```

```
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
```

```
# Evaluate on validation data
val_loss, val_accuracy = model.evaluate(val_gen)
print(f'Validation Loss: {val_loss:.4f}')
print(f'Validation Accuracy: {val_accuracy:.4f}')
```

```
38/38 ━━━━━━━━━━━━━━━━━━━━ 5s 120ms/step - accuracy: 0.6374 - loss: 1.0494
Validation Loss: 1.0736
Validation Accuracy: 0.6270
```

```
# Evaluate on test data
test_loss, test_accuracy = model.evaluate(test_gen)
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` class
  self._warn_if_super_not_called()
37/37 ━━━━━━━━━━━━━━━━━━━━ 363s 10s/step - accuracy: 0.6308 - loss: 1.0597
Test Loss: 1.0868
Test Accuracy: 0.6119
```

```
# Get predictions for validation set
val_predictions = model.predict(val_gen)
val_pred_classes = np.argmax(val_predictions, axis=1)  # Get class indices
val_true_classes = val_gen.classes  # True class labels
```

```
38/38 ━━━━━━━━━━━━━━━━━━━━ 12s 191ms/step
```

```
# Get predictions for test set
test_predictions = model.predict(test_gen)
test_pred_classes = np.argmax(test_predictions, axis=1)
test_true_classes = test_gen.classes
```

```
37/37 ━━━━━━━━━━━━━━━━━━━━ 6s 153ms/step
```

```
# Generate classification report for validation data
val_class_report = classification_report(val_true_classes, val_pred_classes, target_names=val_gen.class_indices.keys())
print("Validation Classification Report:\n", val_class_report)
```

```
Validation Classification Report:
                  precision    recall  f1-score   support

              A+       0.13      0.16      0.14       150
              A-       0.15      0.11      0.13       150
             AB+       0.10      0.09      0.09       150
             AB-       0.13      0.19      0.15       150
              B+       0.08      0.09      0.08       150
              B-       0.13      0.11      0.12       151
              O+       0.13      0.13      0.13       150
              O-       0.14      0.09      0.11       150

        accuracy                           0.12      1201
       macro avg       0.12      0.12      0.12      1201
    weighted avg       0.12      0.12      0.12      1201
```

```
# Generate classification report for test data
test_class_report = classification_report(test_true_classes, test_pred_classes, target_names=test_gen.class_indices.keys())
print("Test Classification Report:\n", test_class_report)
```

```
Test Classification Report:
                  precision    recall  f1-score   support

              A+       0.09      0.12      0.10       145
```

A-	0.10	0.08	0.09	145
AB+	0.13	0.12	0.13	145
AB-	0.18	0.26	0.21	145
B+	0.14	0.13	0.14	145
B-	0.07	0.06	0.06	147
O+	0.08	0.08	0.08	145
O-	0.11	0.07	0.09	145
accuracy			0.12	1162
macro avg	0.11	0.12	0.11	1162
weighted avg	0.11	0.12	0.11	1162