DS340 Final Project

Image Classification of Luggage X-ray Dataset

For this project, we use this dataset from: https://universe.roboflow.com/airport-security-scanning/airport-security-scans-dataset.

As you can see in this data set we have images of X-rays of luggage and it is classified into 5 different types of contraband. Our goal is to create a model that can accurately classify these images into the correct type of contraband. If AI can be accurate enough to detect contraband in luggage it can make traveling much safer and give an extra set of eyes to TSA.

```
from google.colab import drive
import zipfile

zip_path = '/content/drive/MyDrive/ds340/project/data.zip'

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall('/content/drive/MyDrive/ds340/project/')

print(os.listdir('/content/drive/MyDrive/ds340/project/'))
```

Load the data

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

```
!ls '/content/drive/MyDrive/ds340/project/'
```

```
data  data1  data_copy  data.zip  idk  __MACOSX  ogdata  originalData
```

```
!ls '/content/drive/MyDrive/ds340/project/data1/test'
```

```
'Folding Knife'  'Multi-tool Knife'   Scissor  'Straight Knife'  'Utility Knife'
```

```
import os

train_dir = '/content/drive/MyDrive/ds340/project/data1/train'
valid_dir = '/content/drive/MyDrive/ds340/project/data1/valid'
test_dir = '/content/drive/MyDrive/ds340/project/data1/test'
test = '/content/drive/MyDrive/ds340/project/data1/test'

print(os.listdir(test_dir))
print(os.listdir(test))
```

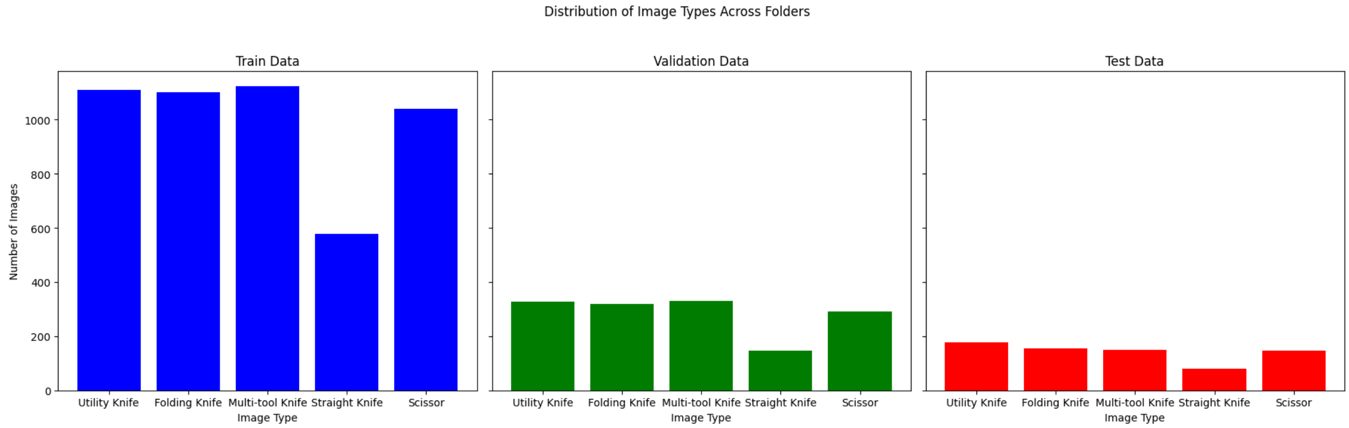
```
['Utility Knife', 'Folding Knife', 'Multi-tool Knife', 'Straight Knife', 'Scisso
['Utility Knife', 'Folding Knife', 'Multi-tool Knife', 'Straight Knife', 'Scisso
```

```
from google.colab import drive
import os

def count_files(directory):
    for subdir in os.listdir(directory):
        subdir_path = os.path.join(directory, subdir)
        if os.path.isdir(subdir_path):
            num_files = len(os.listdir(subdir_path))
            print(f'{subdir} contains {num_files} files')

print("Training Data:")
count_files(train_dir)

print("\nValidation Data:")
count_files(valid_dir)

print("\nTest Data:")
count_files(test_dir)
```

```
Training Data:
Utility Knife contains 1110 files
Folding Knife contains 1102 files
Multi-tool Knife contains 1122 files
Straight Knife contains 578 files
Scissor contains 1039 files

Validation Data:
Utility Knife contains 327 files
Folding Knife contains 319 files
Multi-tool Knife contains 330 files
Straight Knife contains 148 files
Scissor contains 291 files

Test Data:
Utility Knife contains 177 files
Folding Knife contains 155 files
Multi-tool Knife contains 149 files
Straight Knife contains 79 files
Scissor contains 147 files
```

Visualize Data Distribution

```
import os
import matplotlib.pyplot as plt

base_dirs = {
    'train': train_dir,
    'valid': valid_dir,
    'test': test_dir
}

data_summary = {}
total_files = 0

for key, directory in base_dirs.items():
    data_summary[key] = {}
    for subdir in os.listdir(directory):
        subdir_path = os.path.join(directory, subdir)
        if os.path.isdir(subdir_path):
            num_files = len([name for name in os.listdir(subdir_path) if os.path.isf
            data_summary[key][subdir] = num_files
            total_files += num_files

labels = list(data_summary['train'].keys())
train_counts = [data_summary['train'][label] for label in labels]
valid_counts = [data_summary['valid'][label] for label in labels]
test_counts = [data_summary['test'][label] for label in labels]

fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharey=True)
fig.suptitle('Distribution of Image Types Across Folders')

axes[0].bar(labels, train_counts, color='b')
axes[0].set_title('Train Data')
axes[0].set_ylabel('Number of Images')
axes[0].set_xlabel('Image Type')

axes[1].bar(labels, valid_counts, color='g')
axes[1].set_title('Validation Data')
axes[1].set_xlabel('Image Type')

axes[2].bar(labels, test_counts, color='r')
axes[2].set_title('Test Data')
axes[2].set_xlabel('Image Type')

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()

print("Percentage Distribution Across Folders:")
for label in labels:
    total = train_counts[labels.index(label)] + valid_counts[labels.index(label)] +
    print(f"\n{label}:")
    print(f"  Train: {train_counts[labels.index(label)] / total * 100:.2f}%")
    print(f"  Valid: {valid_counts[labels.index(label)] / total * 100:.2f}%")
print(f"  Test: {test_counts[labels.index(label)] / total * 100:.2f}%")
```



```
Percentage Distribution Across Folders:

Utility Knife:
  Train: 68.77%
  Valid: 20.26%
  Test: 10.97%

Folding Knife:
  Train: 69.92%
  Valid: 20.24%
  Test: 9.84%

Multi-tool Knife:
  Train: 70.08%
  Valid: 20.61%
  Test: 9.31%

Straight Knife:
  Train: 71.80%
  Valid: 18.39%
  Test: 9.81%

Scissor:
  Train: 70.35%
  Valid: 19.70%
  Test: 9.95%
```

Double-click (or enter) to edit

Attempt 1 Multiclass Classification using CNN

```
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, D
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
```

```
IMG_WIDTH = 224
IMG_HEIGHT = 224
BATCH_SIZE = 32
```

```
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_dir,
                                                   target_size=(IMG_WIDTH, IMG_HEIGH
                                                   batch_size=BATCH_SIZE,
                                                   class_mode='categorical',
                                                   shuffle=True)
```

```
Found 4951 images belonging to 5 classes.
```

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

```
    Found 1415 images belonging to 5 classes.
```

```
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 : Folding Knife
    1 : Multi-tool Knife
    2 : Scissor
    3 : Straight Knife
    4 : Utility Knife
```

```
x_batch, y_batch = train_generator.next()
print(x_batch.shape)
print(y_batch.shape)
```

```
    (32, 224, 224, 3)
    (32, 5)
```

```
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
Scissor
Straight Knife
Multi-tool Knife
Folding Knife
Utility Knife
Multi-tool Knife
Folding Knife
Folding Knife
Scissor
Folding Knife

Model #1

```
train_datagen = ImageDataGenerator(rescale=1./255)
valid_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical')

validation_generator = valid_datagen.flow_from_directory(
    valid_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical')
```

```
Found 4951 images belonging to 5 classes.
Found 1415 images belonging to 5 classes.
```

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential([

    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),

    Dense(512, activation='relu'),
    Dense(5, activation='softmax')
])

model.compile
```

```
keras.src.engine.training.Model.compile
def compile(optimizer='rmsprop', loss=None, metrics=None, loss_weights=None,
weighted_metrics=None, run_eagerly=None, steps_per_execution=None,
jit_compile=None, pss_evaluation_shards=0, **kwargs)
```

```
      turns on exact evaluation and uses a heuristic for the number of
      shards based on the number of workers. 0, meaning no
      visitation guarantee is provided. NOTE: Custom implementations of
      `Model.test_step` will be ignored when doing exact evaluation.
      Defaults to `0`.
    **kwargs: Arguments supported for backwards compatibility only.
```

```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_15 (Conv2D)          (None, 148, 148, 32)      896       
                                                                 
 max_pooling2d_15 (MaxPooli  (None, 74, 74, 32)        0         
 ng2D)                                                           
                                                                 
 conv2d_16 (Conv2D)          (None, 72, 72, 64)        18496     
                                                                 
 max_pooling2d_16 (MaxPooli  (None, 36, 36, 64)        0         
 ng2D)                                                           
                                                                 
 conv2d_17 (Conv2D)          (None, 34, 34, 128)       73856     
                                                                 
 max_pooling2d_17 (MaxPooli  (None, 17, 17, 128)       0         
 ng2D)                                                           
                                                                 
 flatten_5 (Flatten)         (None, 36992)             0         
                                                                 
 dense_10 (Dense)            (None, 512)               18940416  
                                                                 
 dense_11 (Dense)            (None, 5)                 2565      
                                                                 
=================================================================
Total params: 19036229 (72.62 MB)
Trainable params: 19036229 (72.62 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'
```

```
history = model.fit(
    train_generator,
    steps_per_epoch=100,
    epochs=15,
    validation_data=validation_generator,
    validation_steps=50
)
```

```
Epoch 1/15
100/100 [==============================] - ETA: 0s - loss: 1.6692 - accuracy: 0.
100/100 [==============================] - 85s 762ms/step - loss: 1.6692 - accur
Epoch 2/15
100/100 [==============================] - 57s 570ms/step - loss: 1.5952 - accur
Epoch 3/15
100/100 [==============================] - 58s 574ms/step - loss: 1.5849 - accur
Epoch 4/15
100/100 [==============================] - 61s 603ms/step - loss: 1.5838 - accur
Epoch 5/15
```

```
    100/100 [==============================] - 61s 607ms/step - loss: 1.5910 - accur
    Epoch 6/15
    100/100 [==============================] - 58s 572ms/step - loss: 1.5857 - accur
    Epoch 7/15
    100/100 [==============================] - 65s 645ms/step - loss: 1.5838 - accur
    Epoch 8/15
    100/100 [==============================] - 67s 660ms/step - loss: 1.5791 - accur
    Epoch 9/15
    100/100 [==============================] - 67s 668ms/step - loss: 1.5556 - accur
    Epoch 10/15
    100/100 [==============================] - 67s 669ms/step - loss: 1.5188 - accur
    Epoch 11/15
    100/100 [==============================] - 70s 698ms/step - loss: 1.4477 - accur
    Epoch 12/15
    100/100 [==============================] - 77s 761ms/step - loss: 1.3012 - accur
    Epoch 13/15
    100/100 [==============================] - 76s 755ms/step - loss: 1.0356 - accur
    Epoch 14/15
    100/100 [==============================] - 74s 734ms/step - loss: 0.7531 - accur
    Epoch 15/15
    100/100 [==============================] - 76s 752ms/step - loss: 0.4282 - accur
```

Wow! 0.8627% accuracy! Don't get your hopes up, way too much overfitting:

tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches (in this case, 50 batches). You may need to use the repeat() function when building your dataset.

```
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)

test_loss, test_accuracy = model.evaluate(test_generator)
print(f"Test loss: {test_loss}, Test accuracy: {test_accuracy}")
```

```
    Found 707 images belonging to 5 classes.
    23/23 [==============================] - 11s 485ms/step - loss: 2.7713 - accurac
    Test loss: 2.7713053226470947, Test accuracy: 0.23620933294296265
```

```
train_count = sum(len(files) for _, _, files in os.walk(train_dir))
valid_count = sum(len(files) for _, _, files in os.walk(valid_dir))
test_count = sum(len(files) for _, _, files in os.walk(test_dir))

print(f"Training Images: {train_count}")
print(f"Validation Images: {valid_count}")
print(f"Test Images: {test_count}")
```

```
    Training Images: 4951
    Validation Images: 1415
    Test Images: 707
```

```
batch_size = 32

train_steps_per_epoch = math.ceil(4951 / batch_size)
validation_steps = math.ceil(1415 / batch_size)


train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=batch_size,
    class_mode='categorical'
)

validation_generator = valid_datagen.flow_from_directory(
    valid_dir,
    target_size=(150, 150),
    batch_size=batch_size,
    class_mode='categorical'
)


history = model.fit(
    train_generator,
    steps_per_epoch=train_steps_per_epoch,
    epochs=15,
    validation_data=validation_generator,
    validation_steps=validation_steps
)
```

```
Found 4951 images belonging to 5 classes.
Found 1415 images belonging to 5 classes.
Epoch 1/15
155/155 [==============================] - 123s 776ms/step - loss: 0.2515 - accu
Epoch 2/15
155/155 [==============================] - 122s 786ms/step - loss: 0.0451 - accu
Epoch 3/15
155/155 [==============================] - 121s 777ms/step - loss: 0.0125 - accu
Epoch 4/15
155/155 [==============================] - 128s 827ms/step - loss: 0.0015 - accu
Epoch 5/15
155/155 [==============================] - 130s 837ms/step - loss: 5.5952e-04 -
Epoch 6/15
155/155 [==============================] - ETA: 0s - loss: 3.7106e-04 - accuracy
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-261-b7a1968d4ce6> in <cell line: 22>()
     20 
     21 # Train the model
---> 22 history = model.fit(
     23     train_generator,
     24     steps_per_epoch=train_steps_per_epoch,

13 frames

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/execute.py in
quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     51   try:
     52     ctx.ensure_initialized()
---> 53     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name,
op_name,
     54                                         inputs, attrs, num_outputs)
     55   except core._NotOkStatusException as e:

KeyboardInterrupt: 
```

No changes in overfitting ^

Model 2 (Pooling and Early Stopping)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, B
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(2, 2),
    Dropout(0.2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    BatchNormalization(),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.3),
    Flatten(),
    Dense(512, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(5, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=

history = model.fit(
    train_generator,
    steps_per_epoch=155,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=45,
    callbacks=[early_stopping]
)
```

```
Epoch 1/10
155/155 [==============================] - 117s 743ms/step - loss: 9.9326 - accu
Epoch 2/10
155/155 [==============================] - 112s 724ms/step - loss: 2.6480 - accu
Epoch 3/10
155/155 [==============================] - 111s 718ms/step - loss: 1.8144 - accu
Epoch 4/10
155/155 [==============================] - 113s 726ms/step - loss: 1.6517 - accu
Epoch 5/10
155/155 [==============================] - 111s 714ms/step - loss: 1.6185 - accu
Epoch 6/10
155/155 [==============================] - 110s 710ms/step - loss: 1.6132 - accu
Epoch 7/10
155/155 [==============================] - 109s 704ms/step - loss: 1.6152 - accu
Epoch 8/10
155/155 [==============================] - 111s 717ms/step - loss: 1.6231 - accu
Epoch 9/10
155/155 [==============================] - 116s 744ms/step - loss: 1.6194 - accu
```

```
    Epoch 10/10
    155/155 [==============================] - 129s 833ms/step - loss: 1.6183 - accu
```

Model 3: Adding Batch Normalization

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, B

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(2, 2),
    BatchNormalization(),

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

    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.5),
    BatchNormalization(),

    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(5, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'
```

```
history = model.fit(
    train_generator,
    steps_per_epoch=155,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=45,
    callbacks=[early_stopping]
)
```

```
    Epoch 1/10
    155/155 [==============================] - 132s 841ms/step - loss: 2.2808 - accu
    Epoch 2/10
    155/155 [==============================] - 110s 707ms/step - loss: 1.6092 - accu
    Epoch 3/10
    155/155 [==============================] - 112s 719ms/step - loss: 1.5939 - accu
    Epoch 4/10
    155/155 [==============================] - 113s 729ms/step - loss: 1.5922 - accu
    Epoch 5/10
    155/155 [==============================] - 113s 731ms/step - loss: 1.5906 - accu
    Epoch 6/10
    155/155 [==============================] - 110s 709ms/step - loss: 1.5894 - accu
    Epoch 7/10
    155/155 [==============================] - 109s 703ms/step - loss: 1.5906 - accu
    Epoch 8/10
    155/155 [==============================] - 108s 695ms/step - loss: 1.5798 - accu
```

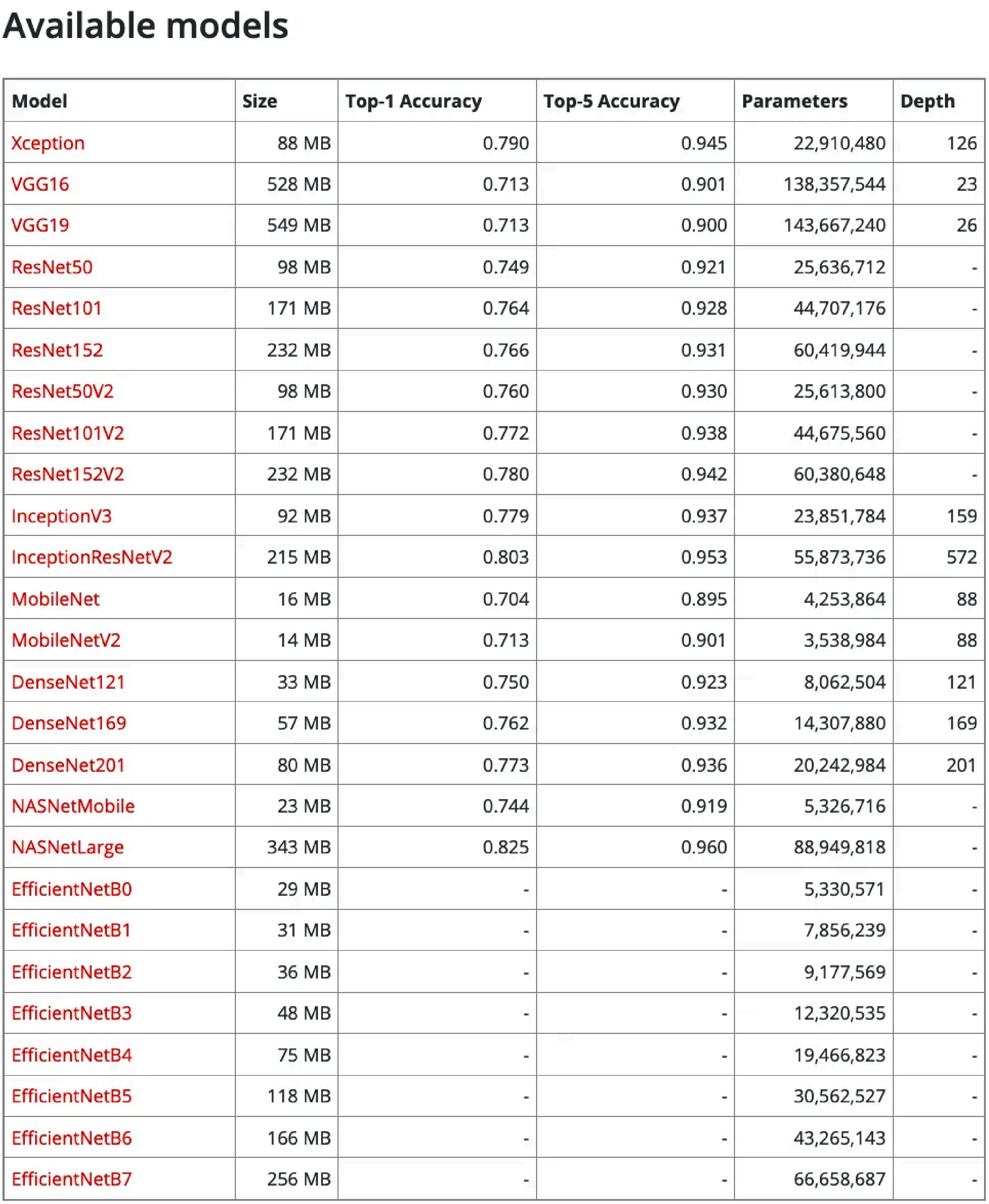
```
test_loss, test_accuracy = model.evaluate(test_generator)
print(f"Test Loss: {test_loss}, Test Accuracy: {test_accuracy}")
```

```
    23/23 [==============================] - 8s 363ms/step - loss: 1.5875 - accuracy
    Test Loss: 1.587489366531372, Test Accuracy: 0.21074964106082916
```

These base models aren't really getting us anywhere. There were many other variations of hyperparameters that I tried but not above as I simply just changed them within the current models and just ran them again.

Attempt 2 Let's Try Transfer Learning

Chart of Different Models



Available models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
EfficientNetB0	29 MB	-	-	5,330,571	-
EfficientNetB1	31 MB	-	-	7,856,239	-
EfficientNetB2	36 MB	-	-	9,177,569	-
EfficientNetB3	48 MB	-	-	12,320,535	-
EfficientNetB4	75 MB	-	-	19,466,823	-
EfficientNetB5	118 MB	-	-	30,562,527	-
EfficientNetB6	166 MB	-	-	43,265,143	-
EfficientNetB7	256 MB	-	-	66,658,687	-

This chart shows us many options that we can proceed with, for the sake of time and available hardware we decided to test out 2, VGG16 and ResNet50.

Chart from: https://medium.com/@blant.jesse/transfer-learning-neur-12df2f55b601

ResNet50

```
from tensorflow.keras.applications import ResNet50

train_datagen = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.resnet.preprocess_input,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

valid_datagen = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.resnet.preprocess_input
)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

validation_generator = valid_datagen.flow_from_directory(
    valid_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

base_model = ResNet50(include_top=False, weights='imagenet', input_shape=(224, 224,

for layer in base_model.layers:
    layer.trainable = False

model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(1024, activation='relu'),
    Dense(5, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'
```

```
Found 4951 images belonging to 5 classes.
Found 1415 images belonging to 5 classes.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
94765736/94765736 [==============================] - 0s 0us/step
```

```
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 resnet50 (Functional)       (None, 7, 7, 2048)        23587712  
                                                                 
 global_average_pooling2d (  (None, 2048)              0         
 GlobalAveragePooling2D)                                         
                                                                 
 dense_18 (Dense)            (None, 1024)              2098176   
                                                                 
 dense_19 (Dense)            (None, 5)                 5125      
                                                                 
=================================================================
Total params: 25691013 (98.00 MB)
Trainable params: 2103301 (8.02 MB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```

```
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size
)
```

```
Epoch 1/10
154/154 [==============================] - 354s 2s/step - loss: 1.8487 - accurac
Epoch 2/10
154/154 [==============================] - 344s 2s/step - loss: 1.5682 - accurac
Epoch 3/10
154/154 [==============================] - 354s 2s/step - loss: 1.5639 - accurac
Epoch 4/10
154/154 [==============================] - 351s 2s/step - loss: 1.5534 - accurac
Epoch 5/10
154/154 [==============================] - 351s 2s/step - loss: 1.5445 - accurac
Epoch 6/10
154/154 [==============================] - 342s 2s/step - loss: 1.5367 - accurac
Epoch 7/10
154/154 [==============================] - 340s 2s/step - loss: 1.5338 - accurac
Epoch 8/10
154/154 [==============================] - 337s 2s/step - loss: 1.5298 - accurac
Epoch 9/10
154/154 [==============================] - 339s 2s/step - loss: 1.5233 - accurac
Epoch 10/10
154/154 [==============================] - 339s 2s/step - loss: 1.5109 - accurac
```

```
test_datagen = ImageDataGenerator(preprocessing_function=tf.keras.applications.resne
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

test_loss, test_accuracy = model.evaluate(test_generator)
print(f"Test Loss: {test_loss}, Test Accuracy: {test_accuracy}")
```

```
Found 707 images belonging to 5 classes.
23/23 [==============================] - 38s 2s/step - loss: 1.5787 - accuracy:
Test Loss: 1.5786525011062622, Test Accuracy: 0.3055162727832794
```

This works better than before, accuracy went up with not much change in loss in comparison to the previous models we tried. Now onto VGG16.

VGG16

```
from tensorflow.keras.applications import VGG16

train_datagen = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.vgg16.preprocess_input,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

valid_datagen = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.vgg16.preprocess_input
)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

validation_generator = valid_datagen.flow_from_directory(
    valid_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

base_model = VGG16(include_top=False, weights='imagenet', input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False

model = Sequential([
    base_model,
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(5, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'
```

```
Found 4951 images belonging to 5 classes.
Found 1415 images belonging to 5 classes.
```

```
model.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten_14 (Flatten)        (None, 25088)             0

 dense_30 (Dense)            (None, 512)               12845568

 dropout_15 (Dropout)        (None, 512)               0

 dense_31 (Dense)            (None, 5)                 2565

=================================================================
Total params: 27562821 (105.14 MB)
Trainable params: 12848133 (49.01 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size
)
```

```
Epoch 1/10
154/154 [==============================] - 964s 6s/step - loss: 10.1694 - accura
Epoch 2/10
154/154 [==============================] - 927s 6s/step - loss: 1.6332 - accurac
Epoch 3/10
154/154 [==============================] - 997s 6s/step - loss: 1.6144 - accurac
Epoch 4/10
 81/154 [==============>...............] - ETA: 5:47 - loss: 1.6150 - accuracy:
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-283-9499a636fc1d> in <cell line: 1>()
----> 1 history = model.fit(
      2     train_generator,
      3     steps_per_epoch=train_generator.samples //
train_generator.batch_size,
      4     epochs=10,
      5     validation_data=validation_generator,

                           10 frames
/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/execute.py in
quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     51   try:
     52     ctx.ensure_initialized()
---> 53     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name,
op_name,
     54                                         inputs, attrs, num_outputs)
     55   except core._NotOkStatusException as e:

KeyboardInterrupt:
```

We stopped running this as we have already run the whole thing in a previous notebook and it is taking too long but here are the results from VGG16 from the previous notebook. Given to the time constraint we chose to continue along with ResNet50

```
Epoch 4/20
124/124 [==============================] - 238s 2s/step - loss: 1.4108 - accuracy: 0.2831 - val_loss: 1.9750 - val_accuracy: 0.2311
Epoch 5/20
124/124 [==============================] - 251s 2s/step - loss: 1.4062 - accuracy: 0.2930 - val_loss: 2.0215 - val_accuracy: 0.2332
Epoch 6/20
124/124 [==============================] - 248s 2s/step - loss: 1.4074 - accuracy: 0.2864 - val_loss: 1.9368 - val_accuracy: 0.2332
Epoch 7/20
124/124 [==============================] - 248s 2s/step - loss: 1.4069 - accuracy: 0.2836 - val_loss: 1.8395 - val_accuracy: 0.2254
Epoch 8/20
124/124 [==============================] - 250s 2s/step - loss: 1.4062 - accuracy: 0.2975 - val_loss: 2.1161 - val_accuracy: 0.2332
Epoch 9/20
124/124 [==============================] - 251s 2s/step - loss: 1.4061 - accuracy: 0.2811 - val_loss: 2.0003 - val_accuracy: 0.2311
Epoch 10/20
124/124 [==============================] - 251s 2s/step - loss: 1.4116 - accuracy: 0.2755 - val_loss: 1.8909 - val_accuracy: 0.2311
Epoch 11/20
124/124 [==============================] - 247s 2s/step - loss: 1.4069 - accuracy: 0.2879 - val_loss: 1.9056 - val_accuracy: 0.2332
Epoch 12/20
124/124 [==============================] - 249s 2s/step - loss: 1.4093 - accuracy: 0.2728 - val_loss: 2.0957 - val_accuracy: 0.2332
Epoch 13/20
124/124 [==============================] - 241s 2s/step - loss: 1.4066 - accuracy: 0.2839 - val_loss: 2.0118 - val_accuracy: 0.2311
Epoch 14/20
124/124 [==============================] - 249s 2s/step - loss: 1.4057 - accuracy: 0.2877 - val_loss: 2.0179 - val_accuracy: 0.2311
Epoch 15/20
124/124 [==============================] - 248s 2s/step - loss: 1.4063 - accuracy: 0.2861 - val_loss: 2.0415 - val_accuracy: 0.2311
Epoch 16/20
124/124 [==============================] - 250s 2s/step - loss: 1.4068 - accuracy: 0.2882 - val_loss: 1.9349 - val_accuracy: 0.2311
Epoch 17/20
124/124 [==============================] - 252s 2s/step - loss: 1.4060 - accuracy: 0.2745 - val_loss: 1.9207 - val_accuracy: 0.2311
Epoch 18/20
124/124 [==============================] - 250s 2s/step - loss: 1.4046 - accuracy: 0.2831 - val_loss: 1.9695 - val_accuracy: 0.2332
Epoch 19/20
124/124 [==============================] - 236s 2s/step - loss: 1.4057 - accuracy: 0.2776 - val_loss: 1.9698 - val_accuracy: 0.2254
Epoch 20/20
```

As you can see there is no benefit to using VGG16 over ResNet50 so we proceeded with ResNet50.

Attempt 3 Redistributing the Data

```
!ls drive/MyDrive/ds340/project/o
```

```
test  train  valid
```

```
import os
import shutil
from sklearn.model_selection import train_test_split

def create_dataset_copy(original_dir, copy_dir):
    """Copies the dataset from the original directory to a new directory."""
    if not os.path.exists(copy_dir):
        shutil.copytree(original_dir, copy_dir)
    else:
        print(f"Copy directory {copy_dir} already exists.")

original_base_dir = '/content/drive/MyDrive/ds340/project/data1'
copy_base_dir = '/content/drive/MyDrive/ds340/project/data_copy'

create_dataset_copy(original_base_dir, copy_base_dir)

def consolidate_images(base_dir, classes, set_names):
    """This function now works on the dataset copy."""
    consolidated_dir = os.path.join(base_dir, 'consolidated')
    os.makedirs(consolidated_dir, exist_ok=True)

    for cls in classes:
        class_dir = os.path.join(consolidated_dir, cls)
        os.makedirs(class_dir, exist_ok=True)
        for set_name in set_names:
            src_dir = os.path.join(base_dir, set_name, cls)
            if os.path.exists(src_dir):
                for file in os.listdir(src_dir):
                    src_file_path = os.path.join(src_dir, file)
                    dst_file_path = os.path.join(class_dir, file)
                    if not os.path.exists(dst_file_path):
                        shutil.copy(src_file_path, dst_file_path)  # Change from mov

classes = ['Folding Knife', 'Multi-tool Knife', 'Straight Knife', 'Utility Knife', '
set_names = ['train', 'valid', 'test']
consolidate_images(copy_base_dir, classes, set_names)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-290-3110820b0a69> in <cell line: 55>()
     53         move_files(test_files, os.path.join(base_dir, 'test', cls))
     54 
---> 55 redistribute_images(copy_base_dir, train_ratio=0.70, val_ratio=0.15)
     56 
     57 # Example to count images in the copied directory

<ipython-input-290-3110820b0a69> in redistribute_images(base_dir, train_ratio, 
val_ratio)
     49         train_files, val_files = train_test_split(train_files, 
test_size=val_ratio/(train_ratio+val_ratio), random_state=42)
     50 
---> 51         move_files(train_files, os.path.join(base_dir, 'train', cls))
     52         move_files(val_files, os.path.join(base_dir, 'valid', cls))
     53         move_files(test_files, os.path.join(base_dir, 'test', cls))

NameError: name 'move_files' is not defined
```

```
import os

def count_images(directory):
    """ Count the number of images in each class directory. """
    class_counts = {}
    for class_dir in os.listdir(directory):
        class_path = os.path.join(directory, class_dir)
        if os.path.isdir(class_path):  # Ensure it's a directory
            count = len(os.listdir(class_path))
            class_counts[class_dir] = count
    return class_counts

train_counts = count_images(os.path.join('/content/drive/MyDrive/ds340/project/data_
val_counts = count_images(os.path.join('/content/drive/MyDrive/ds340/project/data_co
test_counts = count_images(os.path.join('/content/drive/MyDrive/ds340/project/data_c

print("Training set counts:", train_counts)
print("Validation set counts:", val_counts)
print("Test set counts:", test_counts)
```

```
Training set counts: {'Utility Knife': 1110, 'Folding Knife': 1103, 'Multi-tool
Validation set counts: {'Utility Knife': 327, 'Folding Knife': 319, 'Multi-tool
Test set counts: {'Utility Knife': 177, 'Folding Knife': 155, 'Multi-tool Knife'
```

```
import os
import shutil

def consolidate_images(base_dir, classes, set_names):
    consolidated_dir = os.path.join(base_dir, 'consolidated')
    os.makedirs(consolidated_dir, exist_ok=True)

    for cls in classes:
        class_dir = os.path.join(consolidated_dir, cls)
        os.makedirs(class_dir, exist_ok=True)

        for set_name in set_names:
            src_dir = os.path.join(base_dir, set_name, cls)
            if os.path.exists(src_dir):
                for file in os.listdir(src_dir):
                    src_file_path = os.path.join(src_dir, file)
                    dst_file_path = os.path.join(class_dir, file)
                    if not os.path.exists(dst_file_path):
                        shutil.move(src_file_path, dst_file_path)

base_dir = '/content/drive/MyDrive/ds340/project/data_copy'
classes = ['Folding Knife', 'Multi-tool Knife', 'Straight Knife', 'Utility Knife', '
set_names = ['train', 'valid', 'test']
consolidate_images(base_dir, classes, set_names)
```

```
from sklearn.model_selection import train_test_split

def redistribute_images(base_dir, train_ratio=0.60, val_ratio=0.20):
    consolidated_dir = os.path.join(base_dir, 'consolidated')
    classes = os.listdir(consolidated_dir)

    for cls in classes:
        class_dir = os.path.join(consolidated_dir, cls)
        files = [os.path.join(class_dir, f) for f in os.listdir(class_dir)]

        train_files, test_files = train_test_split(files, test_size=1-train_ratio-va
        train_files, val_files = train_test_split(train_files, test_size=val_ratio/(

        def move_files(files, target_dir):
            os.makedirs(target_dir, exist_ok=True)
            for f in files:
                shutil.move(f, target_dir)

        move_files(train_files, os.path.join(base_dir, 'train', cls))
        move_files(val_files, os.path.join(base_dir, 'valid', cls))
        move_files(test_files, os.path.join(base_dir, 'test', cls))

redistribute_images(base_dir, train_ratio=0.70, val_ratio=0.15)
```

```
import os

def count_images(directory):
    """ Count the number of images in each class directory. """
    class_counts = {}
    for class_dir in os.listdir(directory):
        class_path = os.path.join(directory, class_dir)
        if os.path.isdir(class_path):  # Ensure it's a directory
            count = len(os.listdir(class_path))
            class_counts[class_dir] = count
    return class_counts

train_counts = count_images(os.path.join('/content/drive/MyDrive/ds340/project/data_
val_counts = count_images(os.path.join('/content/drive/MyDrive/ds340/project/data_co
test_counts = count_images(os.path.join('/content/drive/MyDrive/ds340/project/data_c

print("Training set counts:", train_counts)
print("Validation set counts:", val_counts)
print("Test set counts:", test_counts)
```

```
Training set counts: {'Utility Knife': 1129, 'Folding Knife': 1102, 'Multi-tool
Validation set counts: {'Utility Knife': 242, 'Folding Knife': 238, 'Multi-tool
Test set counts: {'Utility Knife': 243, 'Folding Knife': 237, 'Multi-tool Knife'
```

Oversample Miniority Class - Data Augment

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

datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

```
import os
import tensorflow as tf

base_dir = '/content/drive/MyDrive/ds340/project/data_copy'
class_name = 'Straight Knife'
train_dir = os.path.join(base_dir, 'train', class_name)
valid_dir = os.path.join(base_dir, 'valid', class_name)
test_dir = os.path.join(base_dir, 'test', class_name)

def augment_images(directory, num_augmented_per_image=5):
    for filename in os.listdir(directory):
        if filename.endswith('.jpg') or filename.endswith('.png'):
            image_path = os.path.join(directory, filename)
            image = tf.keras.preprocessing.image.load_img(image_path)
            image = image.resize((150, 150))
            x = tf.keras.preprocessing.image.img_to_array(image)
            x = x.reshape((1,) + x.shape)

            i = 0
            for batch in datagen.flow(x, batch_size=1, save_to_dir=directory, save_p
                i += 1
                if i >= num_augmented_per_image:
                    break

augment_images(train_dir)
augment_images(valid_dir)
augment_images(test_dir)
```

```
import os

def count_images(directory):
    """ Count the number of images in each class directory. """
    class_counts = {}
    for class_dir in os.listdir(directory):
        class_path = os.path.join(directory, class_dir)
        if os.path.isdir(class_path):  # Ensure it's a directory
            count = len(os.listdir(class_path))
            class_counts[class_dir] = count
    return class_counts

train_counts = count_images(os.path.join('/content/drive/MyDrive/ds340/project/data_
val_counts = count_images(os.path.join('/content/drive/MyDrive/ds340/project/data_co
test_counts = count_images(os.path.join('/content/drive/MyDrive/ds340/project/data_c

print("Training set counts:", train_counts)
print("Validation set counts:", val_counts)
print("Test set counts:", test_counts)
```

```
Training set counts: {'Utility Knife': 1129, 'Folding Knife': 1102, 'Multi-tool
Validation set counts: {'Utility Knife': 242, 'Folding Knife': 238, 'Multi-tool
Test set counts: {'Utility Knife': 243, 'Folding Knife': 237, 'Multi-tool Knife'
```

```
import os
import random

def undersample_directory(directory, target_count):
    """ Randomly remove files from a directory to reduce to target_count. """
    files = [os.path.join(directory, f) for f in os.listdir(directory) if f.endswith
    current_count = len(files)
    if current_count <= target_count:
        print(f"No need to remove files from {directory}, count is already at or bel
        return

    remove_count = current_count - target_count
    files_to_remove = random.sample(files, remove_count)

    for file in files_to_remove:
        os.remove(file)
    print(f"Removed {remove_count} files from {directory}")

base_dir = '/content/drive/MyDrive/ds340/project/data_copy'
train_uk_dir = os.path.join(base_dir, 'train', 'Utility Knife')
valid_uk_dir = os.path.join(base_dir, 'valid', 'Utility Knife')
test_uk_dir = os.path.join(base_dir, 'test', 'Utility Knife')

target_train_count = 700
target_valid_count = 150
target_test_count = 150

undersample_directory(train_uk_dir, target_train_count)
undersample_directory(valid_uk_dir, target_valid_count)
undersample_directory(test_uk_dir, target_test_count)
```

```
Removed 429 files from /content/drive/MyDrive/ds340/project/data_copy/train/Util
Removed 92 files from /content/drive/MyDrive/ds340/project/data_copy/valid/Utili
Removed 93 files from /content/drive/MyDrive/ds340/project/data_copy/test/Utilit
```

```
import os

def count_images(directory):
    """ Count the number of images in each class directory. """
    class_counts = {}
    for class_dir in os.listdir(directory):
        class_path = os.path.join(directory, class_dir)
        if os.path.isdir(class_path):  # Ensure it's a directory
            count = len(os.listdir(class_path))
            class_counts[class_dir] = count
    return class_counts

train_counts = count_images(os.path.join('/content/drive/MyDrive/ds340/project/data_
val_counts = count_images(os.path.join('/content/drive/MyDrive/ds340/project/data_co
test_counts = count_images(os.path.join('/content/drive/MyDrive/ds340/project/data_c

print("Training set counts:", train_counts)
print("Validation set counts:", val_counts)
print("Test set counts:", test_counts)
```

```
Training set counts: {'Utility Knife': 700, 'Folding Knife': 1102, 'Multi-tool K
Validation set counts: {'Utility Knife': 150, 'Folding Knife': 238, 'Multi-tool
Test set counts: {'Utility Knife': 150, 'Folding Knife': 237, 'Multi-tool Knife'
```

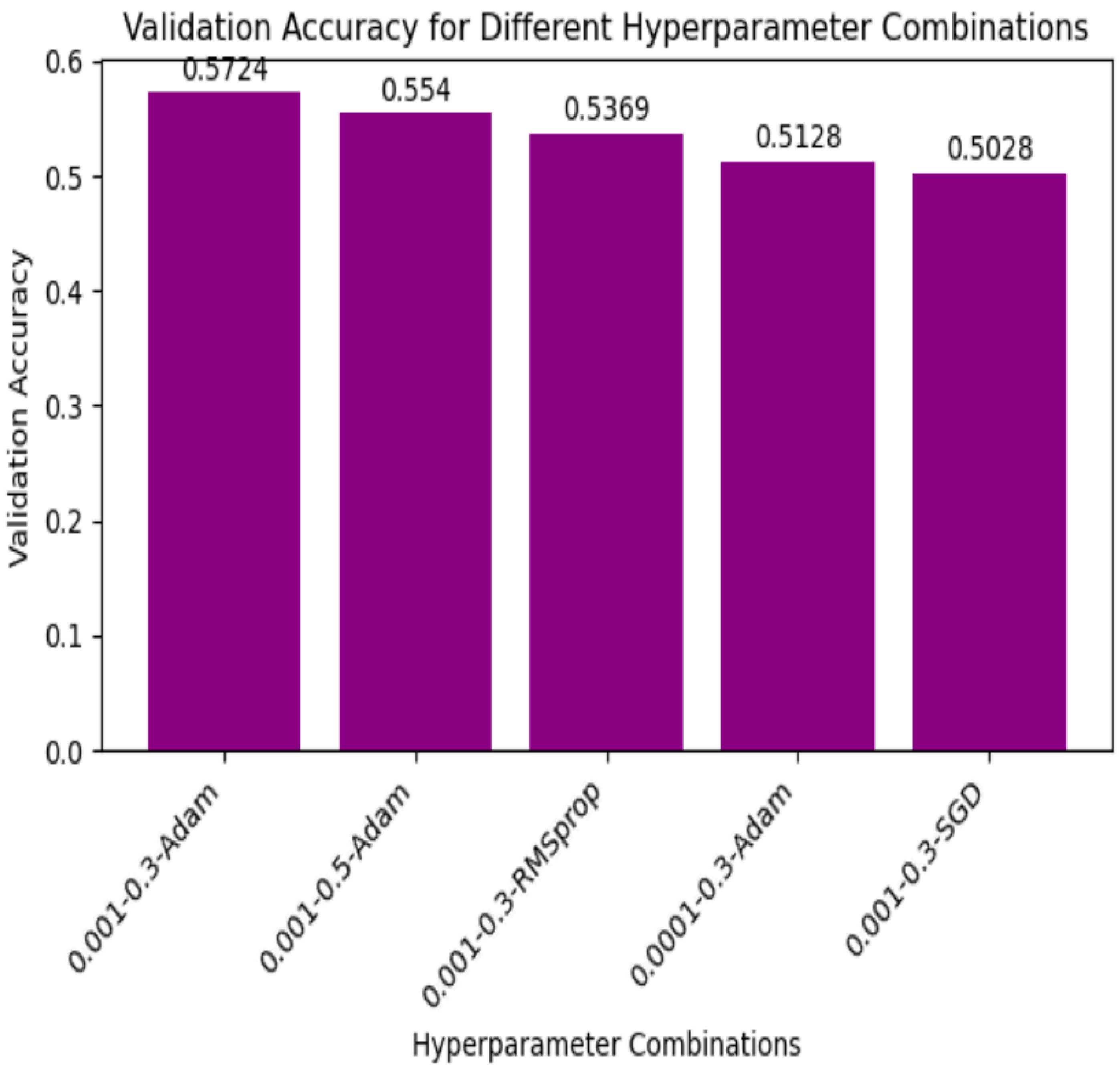
```
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator

def build_model(learning_rate, dropout_rate, optimizer):
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 2

    for layer in base_model.layers:
        layer.trainable = False

    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(dropout_rate)(x)
    predictions = Dense(5, activation='softmax')(x)

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

    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['ac

    return model
learning_rates = [0.001, 0.0001, 0.00001]
dropout_rates = [0.3, 0.5, 0.7]
optimizers = [tf.keras.optimizers.Adam, tf.keras.optimizers.RMSprop, tf.keras.optimi
```

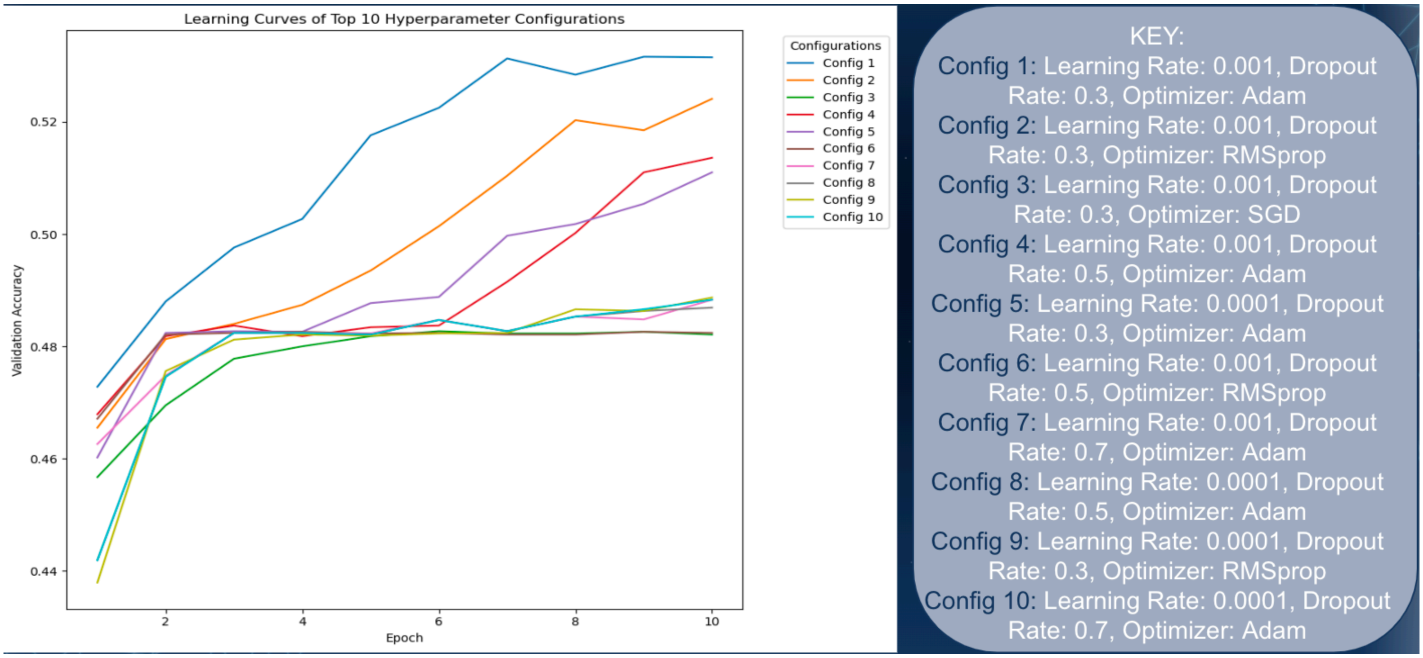
```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam

train_dir = '/content/drive/MyDrive/ds340/project/data_copy/train'
val_dir = '/content/drive/MyDrive/ds340/project/data_copy/valid'
test_dir = '/content/drive/MyDrive/ds340/project/data_copy/test'

# Define image size and batch size
img_height, img_width = 150, 150  # Adjust these dimensions to your specific dataset
batch_size = 32

# Prepare the data generators
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
val_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')

validation_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')

train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

```
Found 6990 images belonging to 5 classes.
Found 1561 images belonging to 5 classes.
Found 1561 images belonging to 5 classes.
```

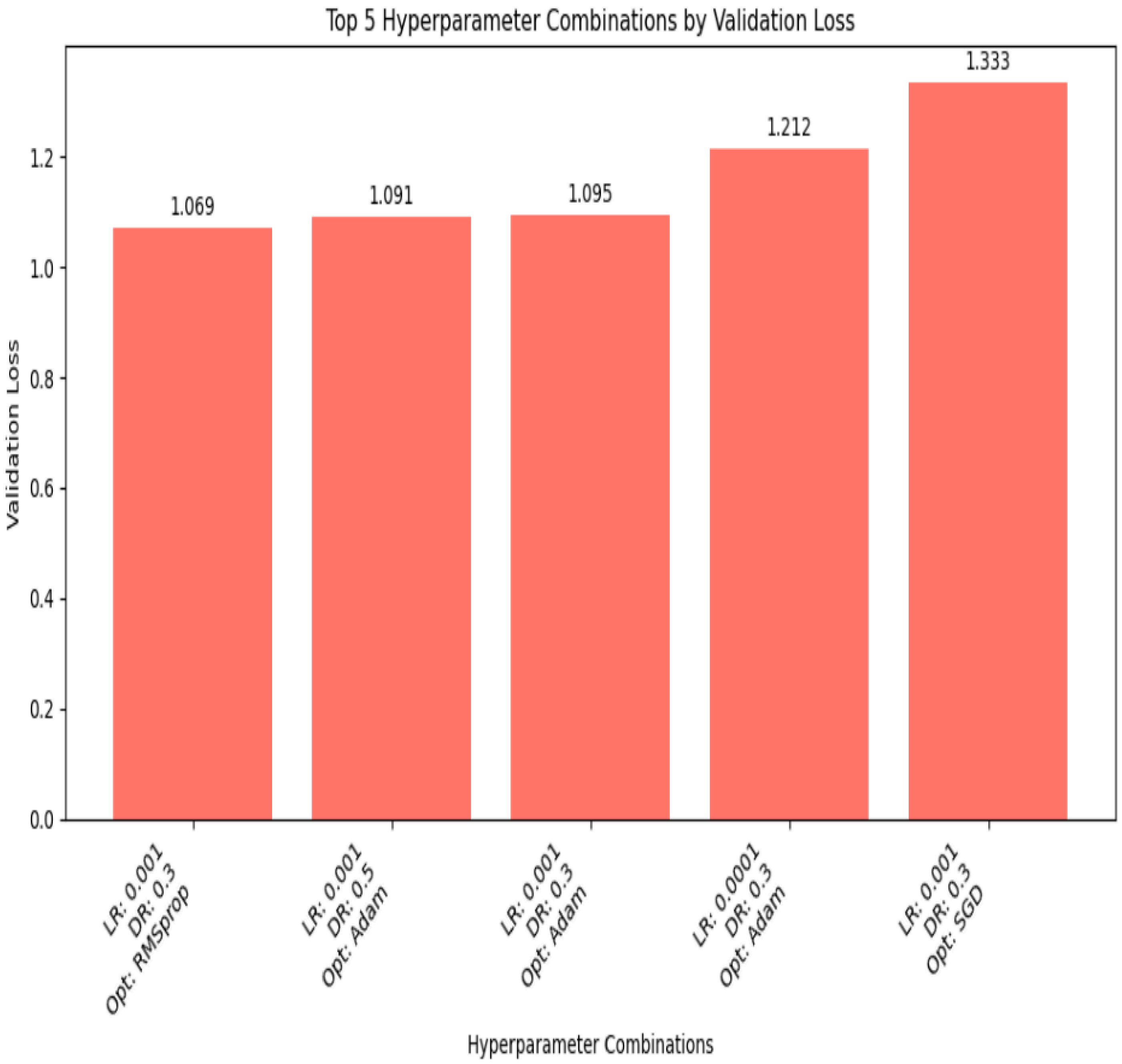
```
for learning_rate in learning_rates:
    for dropout_rate in dropout_rates:
        for optimizer_class in optimizers:
            optimizer = optimizer_class(learning_rate=learning_rate)
            model = build_model(learning_rate, dropout_rate, optimizer)

            history = model.fit(train_generator,
                                steps_per_epoch=train_generator.samples // train_gen
                                epochs=10,
                                validation_data=validation_generator,
                                validation_steps=validation_generator.samples // val

            val_loss, val_acc = model.evaluate(validation_generator,
                                               steps=validation_generator.samples //

            print(f'Learning Rate: {learning_rate}, Dropout Rate: {dropout_rate}, Op
            print(f'Validation Loss: {val_loss}, Validation Accuracy: {val_acc}')
            print('-' * 50)
```

```
Epoch 1/10
196/196 [==============================] - 47s 224ms/step - loss: 2.3437 - accur
Epoch 2/10
196/196 [==============================] - 41s 211ms/step - loss: 2.1287 - accur
Epoch 3/10
196/196 [==============================] - 43s 221ms/step - loss: 1.9778 - accur
Epoch 4/10
196/196 [==============================] - 43s 222ms/step - loss: 1.8564 - accur
Epoch 5/10
196/196 [==============================] - 42s 215ms/step - loss: 1.7765 - accur
Epoch 6/10
196/196 [==============================] - 42s 216ms/step - loss: 1.7261 - accur
Epoch 7/10
196/196 [==============================] - 42s 216ms/step - loss: 1.6948 - accur
Epoch 8/10
196/196 [==============================] - 42s 214ms/step - loss: 1.6814 - accur
Epoch 9/10
196/196 [==============================] - 42s 215ms/step - loss: 1.6621 - accur
Epoch 10/10
196/196 [==============================] - 43s 220ms/step - loss: 1.6378 - accur
44/44 [==============================] - 8s 176ms/step - loss: 1.3672 - accuracy
Learning Rate: 1e-05, Dropout Rate: 0.7, Optimizer: SGD
Validation Loss: 1.367194414138794, Validation Accuracy: 0.5028409361839294
--------------------------------------------------
```


Validation Accuracy for Different Hyperparameter Combinations
Validation Accuracy
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.5724
0.554
0.5369
0.5128
0.5028
0.001-0.3-Adam
0.001-0.5-Adam
0.001-0.3-RMSprop
0.0001-0.3-Adam
0.001-0.3-SGD
Hyperparameter Combinations



KEY:
Config 1: Learning Rate: 0.001, Dropout Rate: 0.3, Optimizer: Adam
Config 2: Learning Rate: 0.001, Dropout Rate: 0.3, Optimizer: RMSprop
Config 3: Learning Rate: 0.001, Dropout Rate: 0.3, Optimizer: SGD
Config 4: Learning Rate: 0.001, Dropout Rate: 0.5, Optimizer: Adam
Config 5: Learning Rate: 0.0001, Dropout Rate: 0.3, Optimizer: Adam
Config 6: Learning Rate: 0.001, Dropout Rate: 0.5, Optimizer: RMSprop
Config 7: Learning Rate: 0.001, Dropout Rate: 0.7, Optimizer: Adam
Config 8: Learning Rate: 0.0001, Dropout Rate: 0.5, Optimizer: Adam
Config 9: Learning Rate: 0.0001, Dropout Rate: 0.3, Optimizer: RMSprop
Config 10: Learning Rate: 0.0001, Dropout Rate: 0.7, Optimizer: Adam


Top 5 Hyperparameter Combinations by Validation Loss
Validation Loss
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.069
1.091
1.095
1.212
1.333
LR: 0.001 DR: 0.3 Opt: RMSprop
LR: 0.001 DR: 0.5 Opt: Adam
LR: 0.001 DR: 0.3 Opt: Adam
LR: 0.0001 DR: 0.3 Opt: Adam
LR: 0.001 DR: 0.3 Opt: SGD
Hyperparameter Combinations