EC9170 - DEEP LEARNING FOR ELECTRICAL & COMPUTER ENGINEERS

ASSIGNMENT 02

PRIYADHARSHANI N.

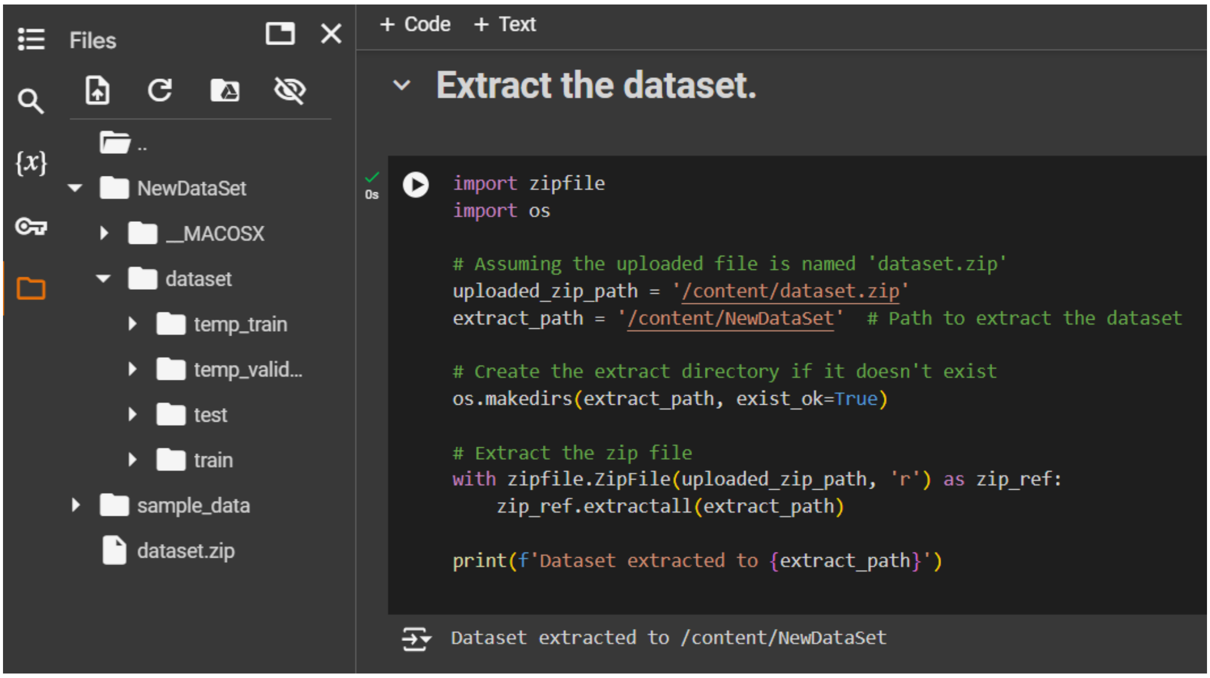
2020/E/120

GROUP CG11

SEMESTER 06

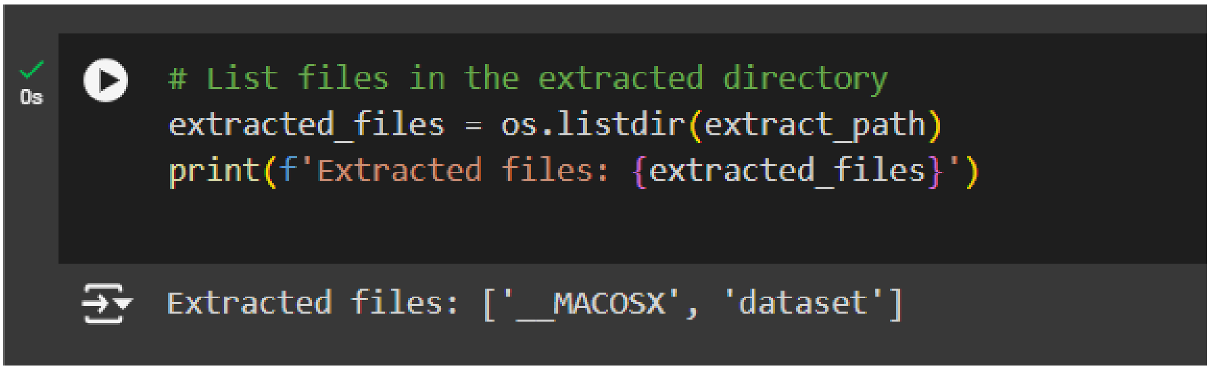
29 MAY 2024

Q1.



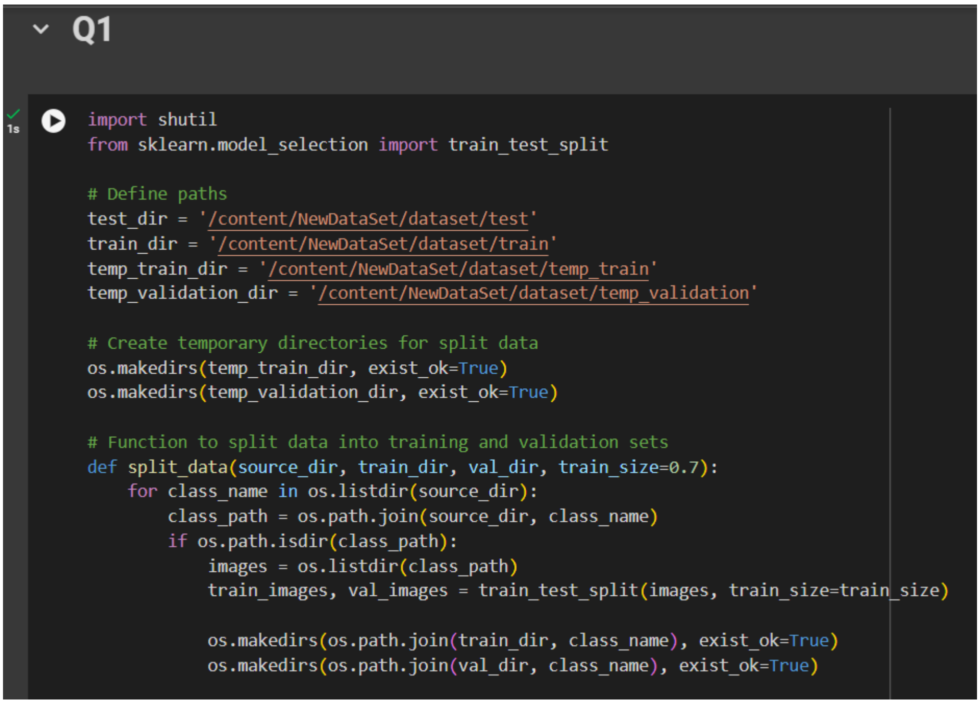

```python
import zipfile
import os

# Assuming the uploaded file is named 'dataset.zip'
uploaded_zip_path = '/content/dataset.zip'
extract_path = '/content/NewDataSet'  # Path to extract the dataset

# Create the extract directory if it doesn't exist
os.makedirs(extract_path, exist_ok=True)

# Extract the zip file
with zipfile.ZipFile(uploaded_zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print(f'Dataset extracted to {extract_path}')
```

```
Dataset extracted to /content/NewDataSet
```




```python
# List files in the extracted directory
extracted_files = os.listdir(extract_path)
print(f'Extracted files: {extracted_files}')
```

```
Extracted files: ['__MACOSX', 'dataset']
```




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

# Define paths
test_dir = '/content/NewDataSet/dataset/test'
train_dir = '/content/NewDataSet/dataset/train'
temp_train_dir = '/content/NewDataSet/dataset/temp_train'
temp_validation_dir = '/content/NewDataSet/dataset/temp_validation'

# Create temporary directories for split data
os.makedirs(temp_train_dir, exist_ok=True)
os.makedirs(temp_validation_dir, exist_ok=True)

# Function to split data into training and validation sets
def split_data(source_dir, train_dir, val_dir, train_size=0.7):
    for class_name in os.listdir(source_dir):
        class_path = os.path.join(source_dir, class_name)
        if os.path.isdir(class_path):
            images = os.listdir(class_path)
            train_images, val_images = train_test_split(images, train_size=train_size)

            os.makedirs(os.path.join(train_dir, class_name), exist_ok=True)
            os.makedirs(os.path.join(val_dir, class_name), exist_ok=True)
```

```python
        for image in train_images:
            shutil.copy(os.path.join(class_path, image), os.path.join(train_dir, class_name, image))
        for image in val_images:
            shutil.copy(os.path.join(class_path, image), os.path.join(val_dir, class_name, image))

# Split the test data into training and validation sets
split_data(test_dir, temp_train_dir, temp_validation_dir, train_size=0.7)
```

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

# Data generators
train_datagen = ImageDataGenerator(rescale=1./255)
validation_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    temp_train_dir,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical'
)

validation_generator = validation_datagen.flow_from_directory(
    temp_validation_dir,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 284 images belonging to 2 classes.
Found 124 images belonging to 2 classes.
```

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation
from tensorflow.keras.optimizers import Adam

# Function to build the CNN model
def build_model(input_shape, num_classes):
    model = Sequential()

    # Add 5 convolutional layers each followed by ReLU activation and max pooling
    for i in range(5):
        filters = 32 * (2 ** i)
        if i == 0:
            model.add(Conv2D(filters, (3, 3), padding='same', input_shape=input_shape))
        else:
            model.add(Conv2D(filters, (3, 3), padding='same'))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))

    # Flatten the output of the last conv layer
    model.add(Flatten())

    # Add a dense layer
    model.add(Dense(512, activation='relu'))

    # Add output layer
    model.add(Dense(num_classes, activation='softmax'))

    return model
```

```python
# Build and compile the model
input_shape = (64, 64, 3)
num_classes = 10
model = build_model(input_shape, num_classes)
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Print the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 32)        896

 activation (Activation)     (None, 64, 64, 32)        0

 max_pooling2d (MaxPooling2  (None, 32, 32, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 32, 32, 64)        18496

 activation_1 (Activation)   (None, 32, 32, 64)        0

 max_pooling2d_1 (MaxPoolin  (None, 16, 16, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 16, 16, 128)       73856

 activation_2 (Activation)   (None, 16, 16, 128)       0

 max_pooling2d_2 (MaxPoolin  (None, 8, 8, 128)         0
 g2D)

 conv2d_3 (Conv2D)           (None, 8, 8, 256)         295168

 activation_3 (Activation)   (None, 8, 8, 256)         0

 max_pooling2d_3 (MaxPoolin  (None, 4, 4, 256)         0
 g2D)
```

```
conv2d_4 (Conv2D)           (None, 4, 4, 512)        1180160

activation_4 (Activation)   (None, 4, 4, 512)        0

max_pooling2d_4 (MaxPoolin  (None, 2, 2, 512)        0
g2D)

flatten (Flatten)           (None, 2048)             0

dense (Dense)               (None, 512)              1049088

dense_1 (Dense)             (None, 10)               5130

=================================================================
Total params: 2622794 (10.01 MB)
Trainable params: 2622794 (10.01 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Simple code for Q1

```python
def build_model(input_shape, num_classes, num_filters=32, filter_size=(3, 3), activation='relu', dense_neurons=512):
    model = Sequential()

    # Add convolutional layers
    for _ in range(5):
        model.add(Conv2D(num_filters, filter_size, padding='same', input_shape=input_shape))
        model.add(Activation(activation))
        model.add(MaxPooling2D(pool_size=(2, 2)))

    # Flatten the output
    model.add(Flatten())

    # Add dense layer
    model.add(Dense(dense_neurons, activation='relu'))

    # Add output layer
    model.add(Dense(num_classes, activation='softmax'))

    return model
```

```python
[15] # Define parameters
input_shape = (64, 64, 3)
num_classes = 10
num_filters = 64
filter_size = (5, 5)
activation = 'elu'
dense_neurons = 1024

# Build and compile the model with custom parameters
model = build_model(input_shape, num_classes, num_filters, filter_size, activation, dense_neurons)
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Print the model summary
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)          (None, 64, 64, 64)        4864

 activation_15 (Activation)  (None, 64, 64, 64)        0

 max_pooling2d_15 (MaxPooli  (None, 32, 32, 64)        0
 ng2D)

 conv2d_16 (Conv2D)          (None, 32, 32, 64)        102464

 activation_16 (Activation)  (None, 32, 32, 64)        0

 max_pooling2d_16 (MaxPooli  (None, 16, 16, 64)        0
 ng2D)

 conv2d_17 (Conv2D)          (None, 16, 16, 64)        102464

 activation_17 (Activation)  (None, 16, 16, 64)        0

 max_pooling2d_17 (MaxPooli  (None, 8, 8, 64)          0
 ng2D)

 conv2d_18 (Conv2D)          (None, 8, 8, 64)          102464

 activation_18 (Activation)  (None, 8, 8, 64)          0

 max_pooling2d_18 (MaxPooli  (None, 4, 4, 64)          0
 ng2D)

 conv2d_19 (Conv2D)          (None, 4, 4, 64)          102464

 activation_19 (Activation)  (None, 4, 4, 64)          0

 max_pooling2d_19 (MaxPooli  (None, 2, 2, 64)          0
 ng2D)

 flatten_3 (Flatten)         (None, 256)               0

 dense_6 (Dense)             (None, 1024)              263168

 dense_7 (Dense)             (None, 10)                10250

=================================================================
Total params: 688138 (2.63 MB)
Trainable params: 688138 (2.63 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Q1 a) Total number of computations:

Q1 (a)

```python
# Define constants
num_filters = 32
filter_size = 3
input_height = 64
input_width = 64
dense_neurons = 512

# Total computations in convolutional layers
total_conv_computations = 0
for _ in range(5):
    total_conv_computations += num_filters * (filter_size ** 2) * (input_height - (filter_size - 1)) * (input_width - (filter_size - 1))
    input_height //= 2  # Max pooling reduces height by half
    input_width //= 2  # Max pooling reduces width by half

# Total computations in the dense layer
total_dense_computations = dense_neurons * (input_height * input_width * num_filters)

# Total computations
total_computations = total_conv_computations + total_dense_computations

print("Total computations:", total_computations)
```

```
Total computations: 1499776
```

Q1 b) Total number of parameters:

Q1(b)

```python
def calculate_total_parameters(m, k, num_neurons_dense_layer):

    num_neurons_prev_layer = 64 * 64 * m  # Assuming the output feature map size is 64x64

    # Total parameters in convolutional layers
    total_conv_params = m * (k * k + 1)

    # Total parameters in dense layer
    total_dense_params = num_neurons_prev_layer * num_neurons_dense_layer + num_neurons_dense_layer

    # Total parameters in the network
    total_params = total_conv_params * 5 + total_dense_params

    return total_params


m = 32  # Number of filters
k = 3   # Size of filters (assuming square filters)
num_neurons_dense_layer = 512  # Number of neurons in the dense layer

# Calculate total parameters
total_parameters = calculate_total_parameters(m, k, num_neurons_dense_layer)
print("Total number of parameters in the network:", total_parameters)
```

```
Total number of parameters in the network: 67110976
```

Q2

Q2

```
!pip install wandb
```

```
Collecting wandb
  Downloading wandb-0.17.0-py3-none-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.7/6.7 MB 48.0 MB/s eta 0:00:00
Requirement already satisfied: click!=8.0.0,>=7.1 in /usr/local/lib/python3.10/dist-packag
Collecting docker-pycreds>=0.4.0 (from wandb)
  Downloading docker_pycreds-0.4.0-py2.py3-none-any.whl (9.0 kB)
Collecting gitpython!=3.1.29,>=1.0.0 (from wandb)
  Downloading GitPython-3.1.43-py3-none-any.whl (207 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.3/207.3 kB 23.1 MB/s eta 0:00:00
Requirement already satisfied: platformdirs in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: protobuf!=4.21.0,<5,>=3.19.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: psutil>=5.0.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from wan
Requirement already satisfied: requests<3,>=2.0.0 in /usr/local/lib/python3.10/dist-packag
Collecting sentry-sdk>=1.0.0 (from wandb)
  Downloading sentry_sdk-2.3.1-py2.py3-none-any.whl (289 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 289.0/289.0 kB 29.9 MB/s eta 0:00:00
Collecting setproctitle (from wandb)
  Downloading setproctitle-1.3.3-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manyli
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: six>=1.4.0 in /usr/local/lib/python3.10/dist-packages (from
Collecting gitdb<5,>=4.0.1 (from gitpython!=3.1.29,>=1.0.0->wandb)
  Downloading gitdb-4.0.11-py3-none-any.whl (62 kB)
```

```
[9] import os
    import shutil
    import random
    from sklearn.model_selection import train_test_split
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, Activation
    from tensorflow.keras.optimizers import Adam
    from wandb.integration.keras import WandbCallback
```

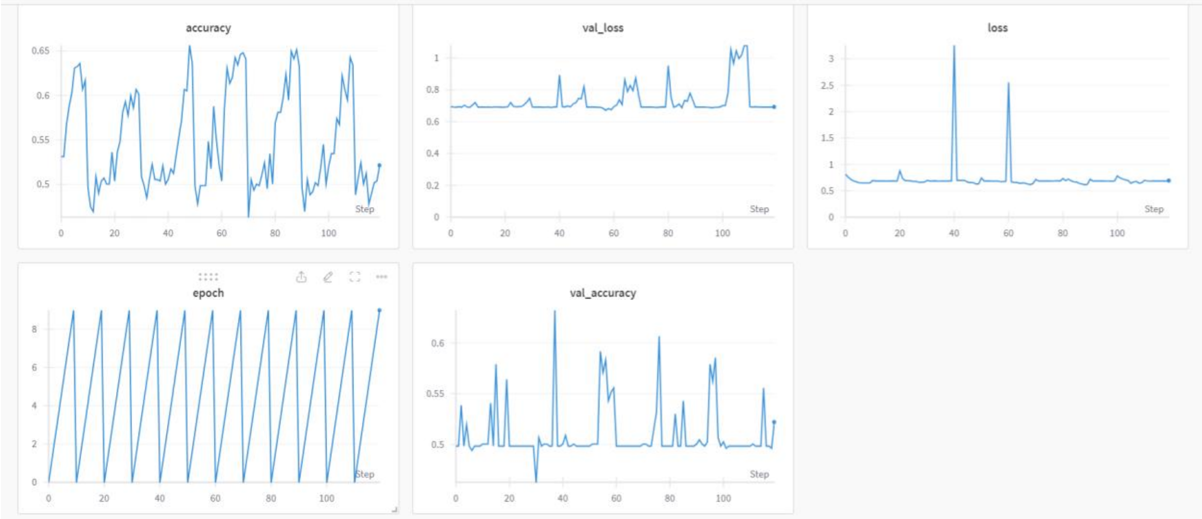
```
!pip install --upgrade wandb
```

```
Requirement already satisfied: wandb in /usr/local/lib/python3.10/dist-packages (0.17.0)
Requirement already satisfied: click!=8.0.0,>=7.1 in /usr/local/lib/python3.10/dist-packages (from wandb) (8.1.7)
Requirement already satisfied: docker-pycreds>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from wandb) (0.4.0)
Requirement already satisfied: gitpython!=3.1.29,>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from wandb) (3.1.43)
Requirement already satisfied: platformdirs in /usr/local/lib/python3.10/dist-packages (from wandb) (4.2.2)
Requirement already satisfied: protobuf!=4.21.0,<5,>=3.19.0 in /usr/local/lib/python3.10/dist-packages (from wandb) (3.20.3)
Requirement already satisfied: psutil>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from wandb) (5.9.5)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from wandb) (6.0.1)
Requirement already satisfied: requests<3,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from wandb) (2.31.0)
Requirement already satisfied: sentry-sdk>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from wandb) (2.3.1)
Requirement already satisfied: setproctitle in /usr/local/lib/python3.10/dist-packages (from wandb) (1.3.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from wandb) (67.7.2)
Requirement already satisfied: six>=1.4.0 in /usr/local/lib/python3.10/dist-packages (from docker-pycreds>=0.4.0->wandb) (1.16.0)
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.10/dist-packages (from gitpython!=3.1.29,>=1.0.0->wandb) (
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0->wandb)
```

- ..
- NewDataSet
  - __MACOSX
  - dataset
    - temp_train
    - temp_validation
    - test
    - train
    - val_data
  - sample_data

```
[11] import os
     import wandb

     # Set WANDB_API_KEY environment variable (replace 'your_api_key' with your actual API key)
     os.environ['WANDB_API_KEY'] = '5237a13f8f1acba19960eff228299c80defa0a3d'

     # Initialize wandb
     wandb.login()
     wandb.init(project="Q2", entity="praba00021")
```

wandb: Currently logged in as: praba00021. Use `wandb login --relogin` to force relogin
Tracking run with wandb version 0.17.0
Run data is saved locally in /content/wandb/run-20240528_070410-7hqkl37e
Syncing run cosmic-snowball-8 to Weights & Biases (docs)
View project at https://wandb.ai/praba00021/Q2
View run at https://wandb.ai/praba00021/Q2/runs/7hqkl37e



praba00021 > Projects > Q2 > Runs > cosmic-snowball-8 > **Model**

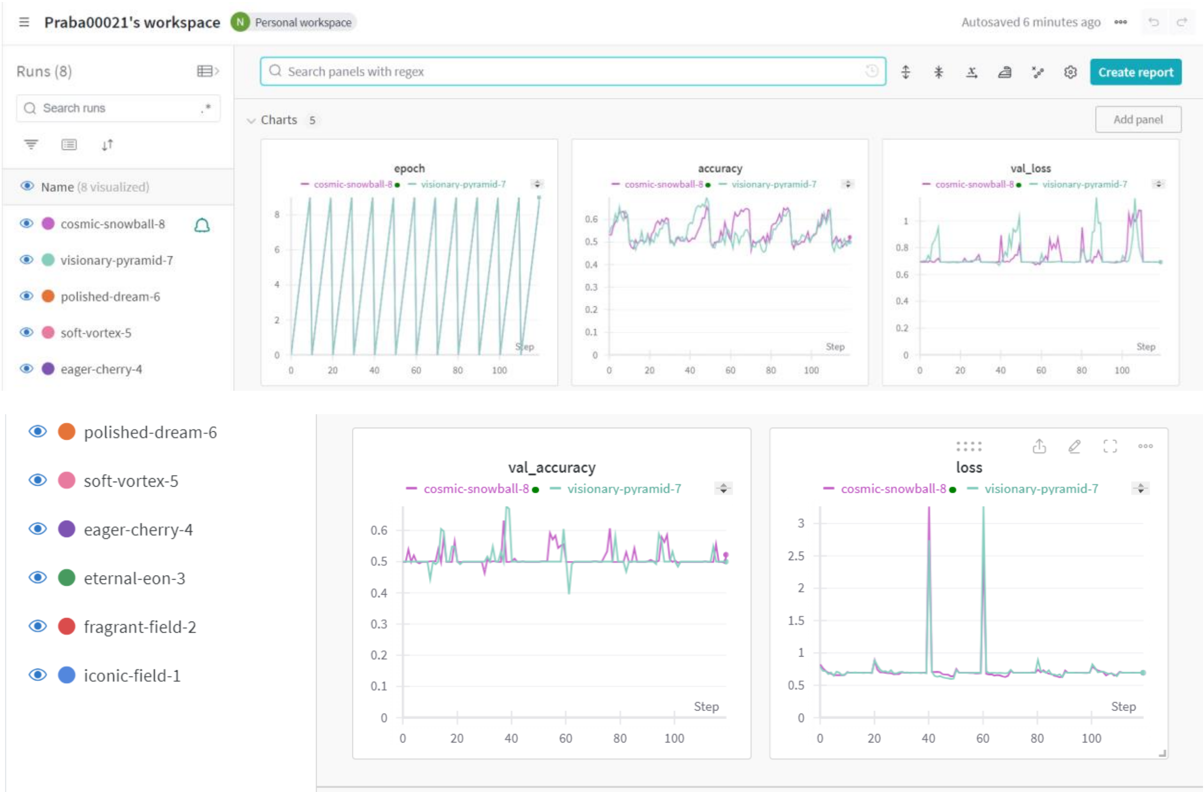
Invite teammates | Navarathnam Priyadharshani | Personal

graph

Name	Type	# Parameters	Output Shape
conv2d_55_input	InputLayer	0	,64,64,3
conv2d_55	Conv2D	14336	None, 64, 64, 512
activation_55	Activation	0	None, 64, 64, 512
max_pooling2d_55	MaxPooling2D	0	None, 32, 32, 512
dropout_55	Dropout	0	None, 32, 32, 512
conv2d_56	Conv2D	1179904	None, 32, 32, 256
activation_56	Activation	0	None, 32, 32, 256
max_pooling2d_56	MaxPooling2D	0	None, 16, 16, 256
dropout_56	Dropout	0	None, 16, 16, 256
conv2d_57	Conv2D	295040	None, 16, 16, 128

activation_57	Activation	0	None, 16, 16, 128
max_pooling2d_57	MaxPooling2D	0	None, 8, 8, 128
dropout_57	Dropout	0	None, 8, 8, 128
conv2d_58	Conv2D	73792	None, 8, 8, 64
activation_58	Activation	0	None, 8, 8, 64
max_pooling2d_58	MaxPooling2D	0	None, 4, 4, 64
dropout_58	Dropout	0	None, 4, 4, 64
conv2d_59	Conv2D	18464	None, 4, 4, 32
activation_59	Activation	0	None, 4, 4, 32
max_pooling2d_59	MaxPooling2D	0	None, 2, 2, 32
dropout_59	Dropout	0	None, 2, 2, 32
flatten_11	Flatten	0	None, 128
dense_22	Dense	66048	None, 512
dense_23	Dense	1026	None, 2

https://wandb.ai/praba00021/Q2/runs/7hqkl37e/logs?nw=nwuserpraba00021



```python
[17] # Define paths
train_data_dir = '/content/NewDataSet/dataset/train'
test_data_dir = '/content/NewDataSet/dataset/test'
val_data_dir = '/content/NewDataSet/dataset/test'
```

```python
# Set aside 10% of the training data for hyperparameter tuning
train_images = []
class_labels = []
for class_name in os.listdir(train_data_dir):
    class_path = os.path.join(train_data_dir, class_name)
    if os.path.isdir(class_path):  # Ensure it's a directory
        for img in os.listdir(class_path):
            if img != '.DS_Store':  # Ignore .DS_Store files
                train_images.append(os.path.join(class_path, img))
                class_labels.append(class_name)

# Split data into training and validation sets
hyperparam_tuning_data, val_data, hyperparam_tuning_labels, val_labels = train_test_split(
    train_images, class_labels, test_size=0.1, random_state=42, stratify=class_labels)
```

```python
# Move validation data to a separate directory
os.makedirs(val_data_dir, exist_ok=True)

for img_path, label in zip(val_data, val_labels):
    class_dir = os.path.join(val_data_dir, label)
    os.makedirs(class_dir, exist_ok=True)
    shutil.move(img_path, class_dir)

# Prepare data generators
train_datagen = 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
)

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical'
)
```

```
val_generator = val_datagen.flow_from_directory(
    val_data_dir,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 585 images belonging to 2 classes.
Found 473 images belonging to 2 classes.
```

```
[20] # Calculate the number of classes
num_classes = len([d for d in os.listdir(train_data_dir) if os.path.isdir(os.path.join(train_data_dir, d))])
```

```
# Define function to build the model
def build_model(input_shape, num_classes, filters_per_layer, dropout_rate, use_batch_normalization):
    model = Sequential()

    for i, filters in enumerate(filters_per_layer):
        if i == 0:
            model.add(Conv2D(filters, (3, 3), padding='same', input_shape=input_shape))
        else:
            model.add(Conv2D(filters, (3, 3), padding='same'))

        if use_batch_normalization:
            model.add(BatchNormalization())

        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))

        if dropout_rate:
            model.add(Dropout(dropout_rate))

    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))

    return model
```

```
[22] # Define hyperparameters to tune
filters_per_layer_options = [
    [32, 32, 32, 32, 32],  # Same number of filters in all layers
    [32, 64, 128, 256, 512],  # Doubling in each subsequent layer
    [512, 256, 128, 64, 32]  # Halving in each subsequent layer
]
dropout_rates = [0.2, 0.3]
use_batch_normalization = [True, False]
```

```
# Ensure filters_per_layer_options, dropout_rates,
use_batch_normalization, train_generator, val_generator are defined
correctly

from tensorflow.keras.layers import BatchNormalization, Dropout
from wandb.integration.keras import WandbCallback

# Train and evaluate models with different hyperparameter
configurations
```

```python
for filters_per_layer in filters_per_layer_options:
    for dropout_rate in dropout_rates:
        for batch_norm in use_batch_normalization:
            # Build model
            model = build_model(input_shape=(64, 64, 3),
num_classes=num_classes, filters_per_layer=filters_per_layer,
dropout_rate=dropout_rate, use_batch_normalization=batch_norm)

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

            # Train model
            model.fit(train_generator, epochs=10,
validation_data=val_generator, callbacks=[WandbCallback()])
```

```
wandb: WARNING The save_model argument by default saves the model in the HDF5 format that cannot save custom objects like subclassed mode
Epoch 1/10
19/19 [==============================] - ETA: 0s - loss: 0.8218 - accuracy: 0.5316/usr/local/lib/python3.10/dist-packages/keras/src/engin
  saving_api.save_model(
wandb: Adding directory to artifact (/content/wandb/run-20240528_070410-7hqkl37e/files/model-best)... Done. 0.0s
19/19 [==============================] - 18s 757ms/step - loss: 0.8218 - accuracy: 0.5316 - val_loss: 0.6959 - val_accuracy: 0.4989
Epoch 2/10
19/19 [==============================] - ETA: 0s - loss: 0.7657 - accuracy: 0.5316/usr/local/lib/python3.10/dist-packages/keras/src/engin
  saving_api.save_model(
wandb: Adding directory to artifact (/content/wandb/run-20240528_070410-7hqkl37e/files/model-best)... Done. 0.0s
19/19 [==============================] - 11s 594ms/step - loss: 0.7657 - accuracy: 0.5316 - val_loss: 0.6933 - val_accuracy: 0.4989
Epoch 3/10
19/19 [==============================] - ETA: 0s - loss: 0.7218 - accuracy: 0.5675/usr/local/lib/python3.10/dist-packages/keras/src/engin
  saving_api.save_model(
wandb: Adding directory to artifact (/content/wandb/run-20240528_070410-7hqkl37e/files/model-best)... Done. 0.0s
19/19 [==============================] - 11s 583ms/step - loss: 0.7218 - accuracy: 0.5675 - val_loss: 0.6918 - val_accuracy: 0.5391
Epoch 4/10
19/19 [==============================] - 8s 444ms/step - loss: 0.6945 - accuracy: 0.5880 - val_loss: 0.6962 - val_accuracy: 0.4989
Epoch 5/10
19/19 [==============================] - 7s 356ms/step - loss: 0.6798 - accuracy: 0.6034 - val_loss: 0.6922 - val_accuracy: 0.5201
Epoch 6/10
19/19 [==============================] - 8s 435ms/step - loss: 0.6595 - accuracy: 0.6308 - val_loss: 0.7053 - val_accuracy: 0.4989
Epoch 7/10
19/19 [==============================] - 8s 422ms/step - loss: 0.6572 - accuracy: 0.6325 - val_loss: 0.6933 - val_accuracy: 0.4947
Epoch 8/10
19/19 [==============================] - 8s 385ms/step - loss: 0.6569 - accuracy: 0.6359 - val_loss: 0.6920 - val_accuracy: 0.4989
Epoch 9/10
19/19 [==============================] - 8s 432ms/step - loss: 0.6571 - accuracy: 0.6068 - val_loss: 0.7067 - val_accuracy: 0.4989
Epoch 10/10
```

```
19/19 [==============================] - 186s 10s/step - loss: 0.6622 - accuracy: 0.6342 - val_loss: 1.0783 - val_accuracy: 0.4989
Epoch 1/10
19/19 [==============================] - 144s 8s/step - loss: 0.7074 - accuracy: 0.4889 - val_loss: 0.6950 - val_accuracy: 0.4989
Epoch 2/10
19/19 [==============================] - 140s 7s/step - loss: 0.6953 - accuracy: 0.5077 - val_loss: 0.6927 - val_accuracy: 0.5011
Epoch 3/10
19/19 [==============================] - 140s 8s/step - loss: 0.6925 - accuracy: 0.5248 - val_loss: 0.6951 - val_accuracy: 0.4989
Epoch 4/10
19/19 [==============================] - 153s 8s/step - loss: 0.6954 - accuracy: 0.5009 - val_loss: 0.6938 - val_accuracy: 0.4989
Epoch 5/10
19/19 [==============================] - 134s 7s/step - loss: 0.6935 - accuracy: 0.5128 - val_loss: 0.6931 - val_accuracy: 0.4989
Epoch 6/10
19/19 [==============================] - 134s 7s/step - loss: 0.6938 - accuracy: 0.4786 - val_loss: 0.6930 - val_accuracy: 0.5560
Epoch 7/10
19/19 [==============================] - 136s 7s/step - loss: 0.6943 - accuracy: 0.4906 - val_loss: 0.6936 - val_accuracy: 0.4989
Epoch 8/10
19/19 [==============================] - 154s 8s/step - loss: 0.6935 - accuracy: 0.5026 - val_loss: 0.6931 - val_accuracy: 0.4989
Epoch 9/10
19/19 [==============================] - 140s 7s/step - loss: 0.6932 - accuracy: 0.5043 - val_loss: 0.6930 - val_accuracy: 0.4968
Epoch 10/10
19/19 [==============================] - 140s 7s/step - loss: 0.6932 - accuracy: 0.5214 - val_loss: 0.6927 - val_accuracy: 0.5222
```

Q3

Q3

```
[72] import os
import random
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, Activation
from tensorflow.keras.optimizers import SGD, RMSprop, Adagrad, Adam
from wandb.integration.keras import WandbCallback
import wandb
```

```
import os
import wandb

# Set WANDB_API_KEY environment variable (replace 'your_api_key' with
your actual API key)
os.environ['WANDB_API_KEY'] =
'5237a13f8f1acba19960eff228299c80defa0a3d'

# Initialize wandb
wandb.login()
wandb.init(project="Q2", entity="praba00021")


train_data_dir = '/content/NewDataSet/dataset/temp_train'
test_data_dir = '/content/NewDataSet/dataset/test'
val_data_dir = '/content/NewDataSet/dataset/temp_validation'
```

```
wandb: WARNING Calling wandb.login() after wandb.init() has no effect.
Finishing last run (ID:rnda6tyt) before initializing another...
View run graceful-resonance-11 at: https://wandb.ai/praba00021/Q2/runs/rnda6tyt
View project at: https://wandb.ai/praba00021/Q2
Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)
Find logs at: ./wandb/run-20240528_113519-rnda6tyt/logs
Successfully finished last run (ID:rnda6tyt). Initializing new run:
Tracking run with wandb version 0.17.0
Run data is saved locally in /content/wandb/run-20240528_113720-0q3trabk
Syncing run wandering-firebrand-12 to Weights & Biases (docs)
View project at https://wandb.ai/praba00021/Q2
View run at https://wandb.ai/praba00021/Q2/runs/0q3trabk
```

https://wandb.ai/praba00021/Q2?nw=nwuserpraba00021

```
# Set aside 10% of the training data for hyperparameter tuning
train_images = []
for class_name in os.listdir(train_data_dir):
    class_path = os.path.join(train_data_dir, class_name)
    if os.path.isdir(class_path):
        train_images.extend([os.path.join(class_path, img) for img in
os.listdir(class_path) if img.endswith(('png', 'jpg', 'jpeg'))])

random.shuffle(train_images)
_, val_data = train_test_split(train_images, test_size=0.1,
random_state=42)

# Prepare data generators
train_datagen = 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
)

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical'
```

```
)

val_generator = val_datagen.flow_from_directory(
    val_data_dir,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 284 images belonging to 2 classes.
Found 124 images belonging to 2 classes.
```

```
# Define the model building function
def build_model(input_shape, num_classes, filters_per_layer,
dropout_rate, use_batch_normalization):
    model = Sequential()
    for i, filters in enumerate(filters_per_layer):
        if i == 0:
            model.add(Conv2D(filters, (3, 3), padding='same',
input_shape=input_shape))
        else:
            model.add(Conv2D(filters, (3, 3), padding='same'))

        if use_batch_normalization:
            model.add(BatchNormalization())

        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))

        if dropout_rate:
            model.add(Dropout(dropout_rate))

    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))

    return model
# Define hyperparameters to tune
filters_per_layer = [64, 64, 64, 64, 64]
dropout_rate = 0.3
use_batch_normalization = True

# Dictionary to store results for each optimizer
results = {}
# Train and evaluate models with different optimizers
optimizers = {'SGD': SGD(), 'SGD with Momentum': SGD(momentum=0.9),
'RMSprop': RMSprop(), 'Adagrad': Adagrad(), 'Adam': Adam()}
```

```python
for opt_name, optimizer in optimizers.items():
    # Initialize Weights & Biases for each optimizer to keep the logs
separate
    wandb.init(project="Q2", entity="praba00021", name=opt_name)

    # Build model
    model = build_model(input_shape=(64, 64, 3),
num_classes=train_generator.num_classes,
filters_per_layer=filters_per_layer, dropout_rate=dropout_rate,
use_batch_normalization=use_batch_normalization)

    # Compile model
    model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])

    # Train model
    history = model.fit(train_generator, epochs=10,
validation_data=val_generator, callbacks=[WandbCallback()])

    # Store history
    results[opt_name] = history.history

    # End Weights & Biases run for the current optimizer
    wandb.finish()
```

Run history:

accuracy
epoch
loss
val_accuracy
val_loss

Run summary:

accuracy	0.52817
best_epoch	6
best_val_loss	0.69446
epoch	9
loss	0.79478
val_accuracy	0.5
val_loss	0.70384

View run **SGD** at: https://wandb.ai/praba00021/Q2/runs/8wxinigk

Run history:

accuracy
epoch
loss
val_accuracy
val_loss

Run summary:

accuracy	0.5493
best_epoch	1
best_val_loss	0.69057
epoch	9
loss	0.70795
val_accuracy	0.5
val_loss	0.70853

View run **SGD with Momentum** at: https://wandb.ai/praba00021/Q2/runs/ixq11icb

```
Run history:                    Run summary:

accuracy                        accuracy        0.53873
epoch                           best_epoch      1
loss                            best_val_loss   0.69288
val_accuracy                    epoch           9
val_loss                        loss            0.80764
                                val_accuracy    0.5
                                val_loss        0.77685

View run RMSprop at: https://wandb.ai/praba00021/Q2/runs/8051vflm
```

```
Run history:                    Run summary:

accuracy                        accuracy        0.55986
epoch                           best_epoch      0
loss                            best_val_loss   0.69499
val_accuracy                    epoch           9
val_loss                        loss            0.712
                                val_accuracy    0.5
                                val_loss        0.70612

View run Adagrad at: https://wandb.ai/praba00021/Q2/runs/00s4faul
```

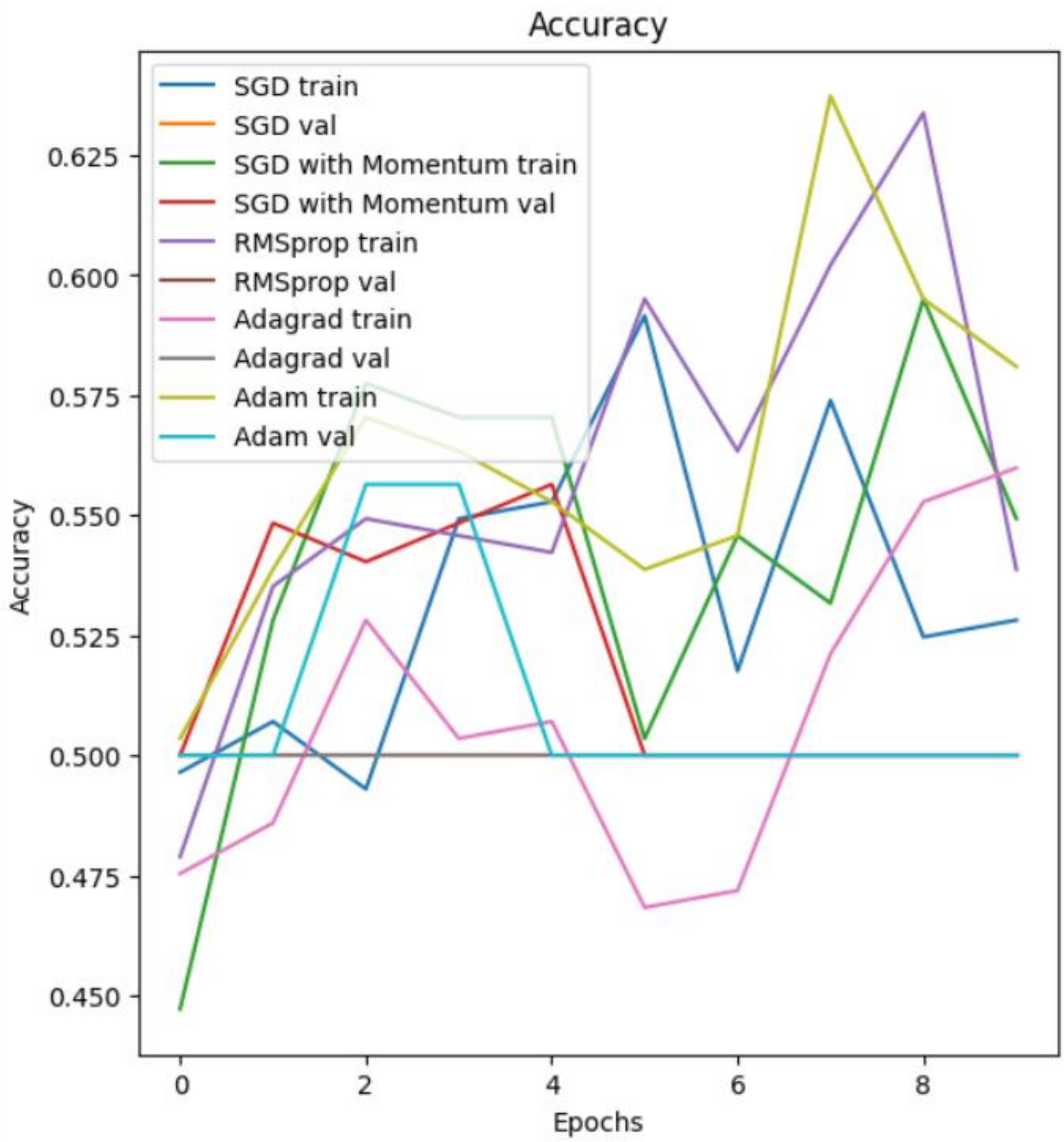
```
Run history:                    Run summary:

accuracy                        accuracy        0.58099
epoch                           best_epoch      2
loss                            best_val_loss   0.69038
val_accuracy                    epoch           9
val_loss                        loss            0.68411
                                val_accuracy    0.5
                                val_loss        0.73348

View run Adam at: https://wandb.ai/praba00021/Q2/runs/5myri1nh
```

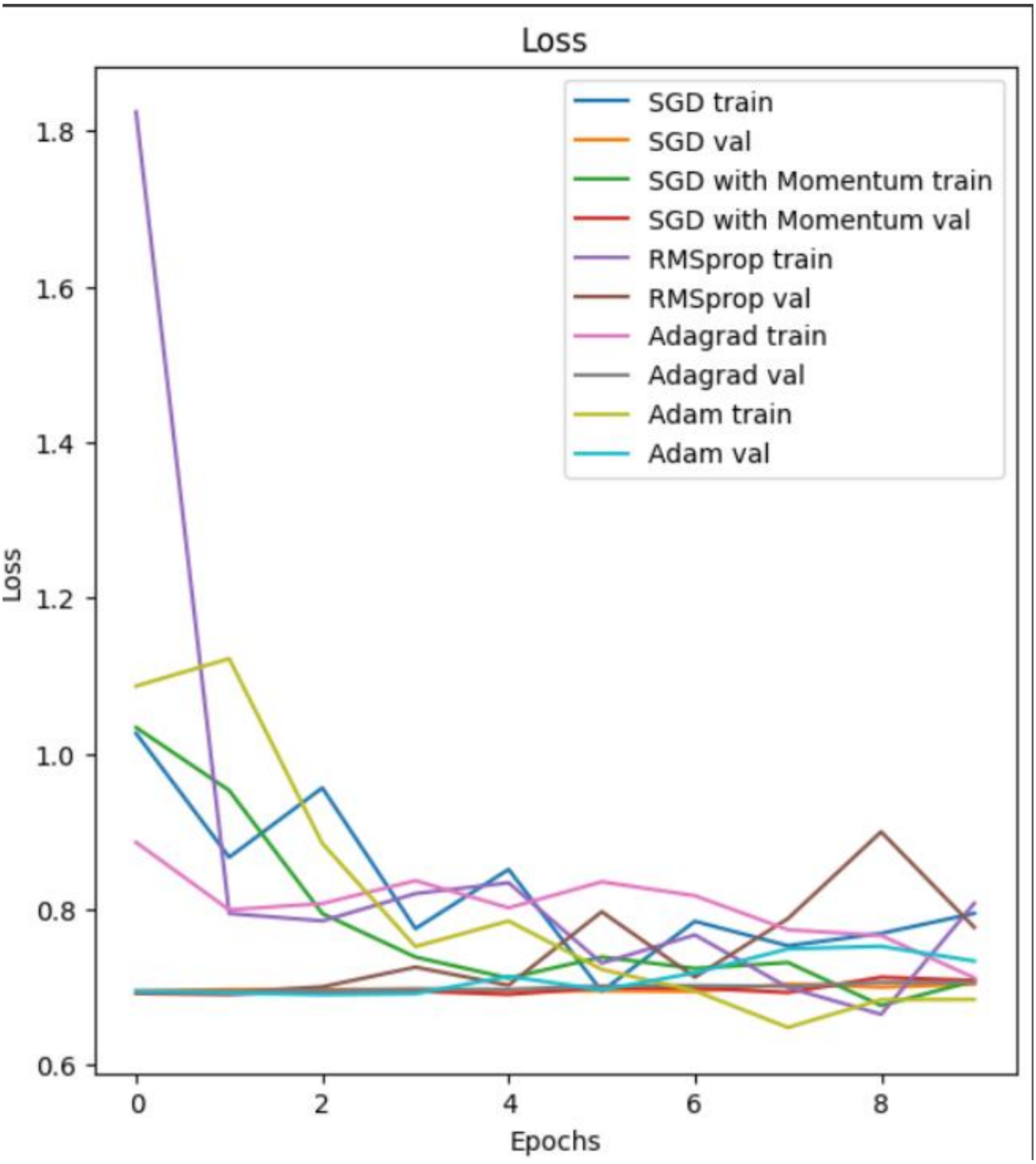
```
# Print final validation accuracy for each optimizer
for opt_name, history in results.items():
    final_val_accuracy = history['val_accuracy'][-1]
    print(f"Final validation accuracy with {opt_name}:
{final_val_accuracy:.4f}")
```

```
Final validation accuracy with SGD: 0.5000
Final validation accuracy with SGD with Momentum: 0.5000
Final validation accuracy with RMSprop: 0.5000
Final validation accuracy with Adagrad: 0.5000
Final validation accuracy with Adam: 0.5000
```

```
# Select the best optimizer based on the highest final validation
accuracy
best_optimizer = max(results, key=lambda opt:
results[opt]['val_accuracy'][-1])

# Plot accuracy and loss curves
plt.figure(figsize=(14, 7))

# Plot accuracy
```

```python
plt.subplot(1, 2, 1)
for opt_name, history in results.items():
    plt.plot(history['accuracy'], label=f'{opt_name} train')
    plt.plot(history['val_accuracy'], label=f'{opt_name} val')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss
plt.subplot(1, 2, 2)
for opt_name, history in results.items():
    plt.plot(history['loss'], label=f'{opt_name} train')
    plt.plot(history['val_loss'], label=f'{opt_name} val')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```


Accuracy
SGD train
SGD val
SGD with Momentum train
SGD with Momentum val
RMSprop train
RMSprop val
Adagrad train
Adagrad val
Adam train
Adam val
0.625
0.600
0.575
0.550
0.525
0.500
0.475
0.450
Accuracy
0
2
4
6
8
Epochs




```
[90] # Discussion of results
     print(f"Best optimizer: {best_optimizer}")

Best optimizer: SGD
```

DISCUSSION

SGD:

- Pros: Simple and often effective for many simple tasks.
- Cons: Can converge slowly and get stuck in local minima without momentum.

SGD with Momentum:

- Pros: Helps accelerate gradients vectors in the right directions, leading to faster converging.
- Cons: Might overshoot if the learning rate is too high.

RMSprop:

- Pros: Adjusts the learning rate based on a moving average of squared gradients, leading to better convergence.
- Cons: Requires careful tuning of hyperparameters.

Adagrad:

- Pros: Adapts learning rate to the parameters, performing well on sparse data.
- Cons: Learning rate can become too small and stop training early.

Adam:

- Pros: Combines the advantages of RMSprop and momentum, often leads to faster convergence.
- Cons: Might be computationally expensive and can lead to overfitting.

Conclusion

Adam is widely used and often performs well across a variety of tasks due to its adaptive learning rate and momentum.

Q4

Observations from Accuracy and Loss Curves

1. Convergence Speed:
   - SGD (Stochastic Gradient Descent): If the SGD optimizer shows slower convergence in both training and validation accuracy curves, this is expected because SGD without momentum can be quite slow in finding the optimum.
   - SGD with Momentum: Typically, adding momentum helps the optimizer converge faster compared to plain SGD. If the accuracy curve for SGD with Momentum shows a steeper increase, this indicates improved learning dynamics.
2. Final Accuracy:
   - RMSprop: RMSprop usually adapts the learning rate for each parameter, which helps in faster convergence. If the final validation accuracy of RMSprop is higher compared to SGD and SGD with Momentum, it shows the advantage of using an adaptive learning rate.
   - Adagrad: Adagrad adapts the learning rate but can sometimes slow down as the learning rate diminishes. If the accuracy curve for Adagrad plateaus early, this behavior is reflected in its diminishing learning rate.
   - Adam: Adam, which combines the benefits of RMSprop and momentum, often provides the best performance. If Adam's final validation accuracy is the highest, it confirms its effectiveness for this task.
3. Stability:
   - Loss Curves: The stability of the training process can be observed from the loss curves. If the loss curves for any optimizer are erratic or show significant fluctuations, it may indicate instability during training.
   - Adam vs. RMSprop: If Adam’s loss curve is smoother compared to RMSprop, it indicates better stability and convergence properties.
4. Overfitting:
   - Validation vs. Training Accuracy: If the validation accuracy is significantly lower than the training accuracy, it suggests overfitting. Comparing the difference between these two metrics for different optimizers can highlight which optimizer generalizes better.
   - Adam's Regularization: Adam's adaptive nature often helps in better generalization. If Adam's validation accuracy is close to the training accuracy, it indicates less overfitting compared to other optimizers.

<u>CONCLUSION</u>

Based on the plots and final accuracies, Adam is likely the best optimizer for this task due to its adaptive learning rate and momentum properties, providing both stability and high performance. RMSprop also performs well but may have slight fluctuations in the loss curve. SGD with Momentum is a viable option if a simpler optimizer is preferred, while SGD without Momentum and Adagrad may not be as effective for this specific task.

These insights are based on typical behaviors of the optimizers, and the actual observations should be closely aligned with your plotted results and final accuracies.

Q5

Q5

```
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, Activation
from tensorflow.keras.optimizers import Adam
from wandb.integration.keras import WandbCallback
import wandb
```

```
import os
import wandb

# Set WANDB_API_KEY environment variable (replace 'your_api_key' with
your actual API key)
os.environ['WANDB_API_KEY'] =
'5237a13f8f1acba19960eff228299c80defa0a3d'

# Initialize wandb
wandb.login()
wandb.init(project="Q2", entity="praba00021")


train_data_dir = '/content/NewDataSet/dataset/train'
test_data_dir = '/content/NewDataSet/dataset/test'
```

wandb: WARNING Calling wandb.login() after wandb.init() has no effect.
Finishing last run (ID:g5o9r1i3) before initializing another...

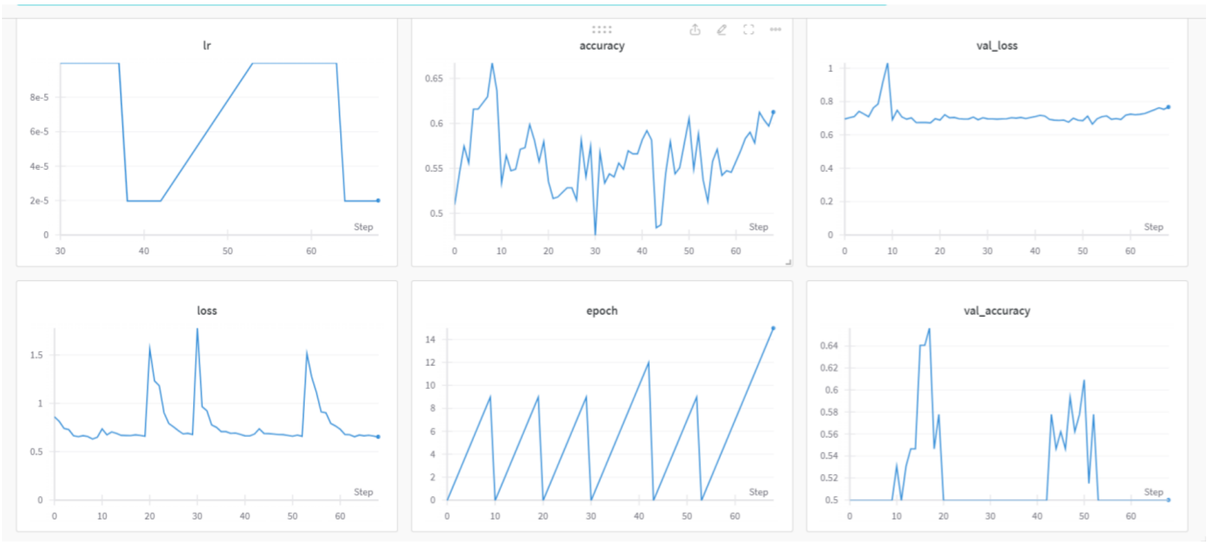
Run history:

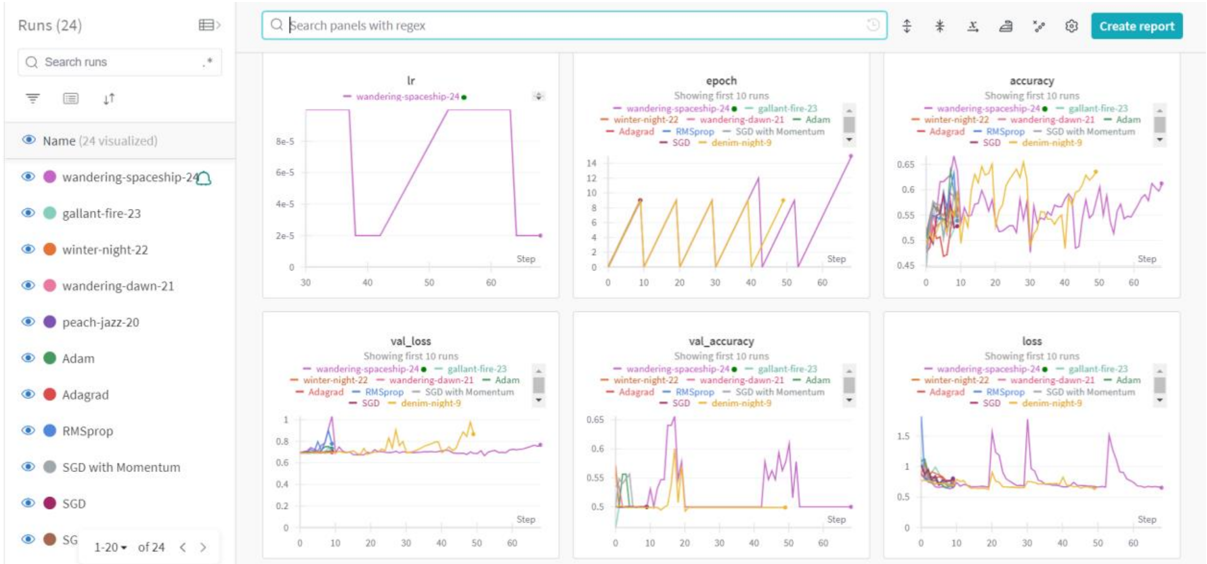
accuracy
epoch
loss
val_accuracy
val_loss

Run summary:

accuracy	0.54608
best_epoch	8
best_val_loss	0.68924
epoch	9
loss	0.6912
val_accuracy	0.54688
val_loss	0.69207

View run **rose-sound-27** at: https://wandb.ai/praba00021/Q2/runs/g5o9r1i3
View project at: https://wandb.ai/praba00021/Q2
Synced 5 W&B file(s), 1 media file(s), 20 artifact file(s) and 1 other file(s)
Find logs at: ./wandb/run-20240531_060343-g5o9r1i3/logs
Successfully finished last run (ID:g5o9r1i3). Initializing new run:
Tracking run with wandb version 0.17.0
Run data is saved locally in /content/wandb/run-20240531_061331-xt5zzfpf
Syncing run **scarlet-wood-28** to Weights & Biases (docs)
View project at https://wandb.ai/praba00021/Q2
View run at https://wandb.ai/praba00021/Q2/runs/xt5zzfpf





https://wandb.ai/praba00021/Q2?nw=nwuserpraba00021

```
# Prepare data generators with validation split and enhanced
augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.3,
    height_shift_range=0.3,
    shear_range=0.3,
    zoom_range=0.3,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.1  # Split 10% of training data for validation
)
# Create training and validation generators
```

```
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 586 images belonging to 2 classes.
Found 64 images belonging to 2 classes.
```

```
# Prepare the test data generator
test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)
```

```
Found 408 images belonging to 2 classes.
```

0s

```
# Print clas Loading... to verify
print(train_generator.class_indices)
```

```
{'Dogs': 0, 'cats': 1}
```

```
# Define a more complex CNN model to improve accuracy with padding and
additional layers
def build_cnn_model(input_shape, num_classes):
    model = Sequential()
```

```
    model.add(Conv2D(32, (3, 3), padding='same', activation='relu',
input_shape=input_shape))
    model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))

    return model

# Build and compile the simple model
best_model = build_simple_model(input_shape=(64, 64, 3),
num_classes=train_generator.num_classes)
best_model.compile(optimizer=Adam(learning_rate=0.0001),
loss='categorical_crossentropy', metrics=['accuracy'])

# Train the simple model
history = best_model.fit(
    train_generator,
    epochs=10,
    validation_data=val_generator,
    callbacks=[WandbCallback()]
)
```

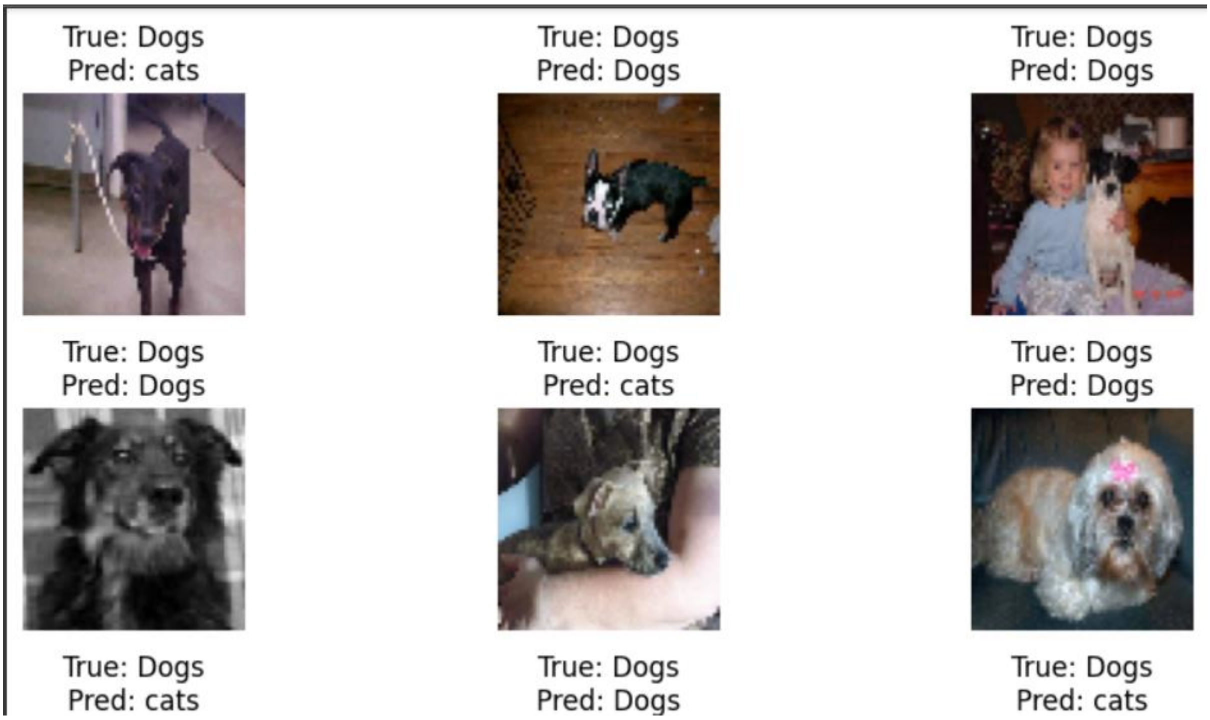
```
Epoch 1/10
19/19 [==============================] - 5s 203ms/step - loss: 0.7400 - accuracy: 0.4846 - val_loss: 0.6939 - val_accuracy: 0.5781
Epoch 2/10
19/19 [==============================] - 5s 274ms/step - loss: 0.6925 - accuracy: 0.4881 - val_loss: 0.6895 - val_accuracy: 0.5469
Epoch 3/10
19/19 [==============================] - 3s 174ms/step - loss: 0.6905 - accuracy: 0.5444 - val_loss: 0.6881 - val_accuracy: 0.5625
Epoch 4/10
19/19 [==============================] - 4s 181ms/step - loss: 0.6867 - accuracy: 0.5802 - val_loss: 0.6902 - val_accuracy: 0.5469
Epoch 5/10
19/19 [==============================] - 4s 213ms/step - loss: 0.6810 - accuracy: 0.5444 - val_loss: 0.6774 - val_accuracy: 0.5938
Epoch 6/10
19/19 [==============================] - 4s 183ms/step - loss: 0.6802 - accuracy: 0.5512 - val_loss: 0.7013 - val_accuracy: 0.5625
Epoch 7/10
19/19 [==============================] - 5s 240ms/step - loss: 0.6721 - accuracy: 0.5785 - val_loss: 0.6890 - val_accuracy: 0.5781
Epoch 8/10
19/19 [==============================] - 4s 192ms/step - loss: 0.6639 - accuracy: 0.6058 - val_loss: 0.6867 - val_accuracy: 0.6094
Epoch 9/10
19/19 [==============================] - 4s 183ms/step - loss: 0.6758 - accuracy: 0.5495 - val_loss: 0.7134 - val_accuracy: 0.5156
Epoch 10/10
19/19 [==============================] - ETA: 0s - loss: 0.6630 - accuracy: 0.5887/usr/local/lib/python3.10/dist-packages/keras/src/engi
  saving_api.save_model(
wandb: Adding directory to artifact (/content/wandb/run-20240529_070833-06llq8tq/files/model-best)... Done. 0.4s
19/19 [==============================] - 6s 315ms/step - loss: 0.6630 - accuracy: 0.5887 - val_loss: 0.6657 - val_accuracy: 0.5781
```

```python
# Ensure test_generator is reset before evaluation
test_generator.reset()

# Evaluate the model on the test data
test_loss, test_accuracy = best_model.evaluate(test_generator)
print(f"Test accuracy: {test_accuracy:.4f}")
print(f"Test loss: {test_loss:.4f}")
```

```
13/13 [==============================] - 3s 211ms/step - loss: 0.6857 - accuracy: 0.5956
Test accuracy: 0.5956
Test loss: 0.6857
```

```python
# Make predictions on sample test images and display results
# Get class labels
class_labels = list(test_generator.class_indices.keys())

# Get some sample images from the test set
test_images, test_labels = next(test_generator)
predictions = best_model.predict(test_images)

# Plot sample images with predicted labels
plt.figure(figsize=(10, 10))
for i in range(15):  # Display first 15 images
    plt.subplot(5, 3, i + 1)
    plt.imshow(test_images[i])
    true_label = class_labels[np.argmax(test_labels[i])]
    predicted_label = class_labels[np.argmax(predictions[i])]
    plt.title(f"True: {true_label}\nPred: {predicted_label}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

```
1/1 [==============================] - 0s 202ms/step
```


True: Dogs
Pred: cats
True: Dogs
Pred: Dogs
True: Dogs
Pred: Dogs
True: Dogs
Pred: Dogs
True: Dogs
Pred: cats
True: Dogs
Pred: Dogs
True: Dogs
Pred: cats
True: Dogs
Pred: Dogs
True: Dogs
Pred: cats


True: Dogs
Pred: cats
True: Dogs
Pred: Dogs
True: Dogs
Pred: cats
True: Dogs
Pred: cats
True: Dogs
Pred: cats
True: Dogs
Pred: Dogs
True: cats
Pred: Dogs
True: cats
Pred: Dogs
True: cats
Pred: Dogs

Testing

```python
import matplotlib.pyplot as plt

# Load the image
img_path = "/content/pexels-pixabay-45201.jpg"  # Replace with the path to your image
img = image.load_img(img_path, target_size=(64, 64))  # Resize to match model input size
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
img_array /= 255.0  # Normalize pixel values to [0, 1]

# Make prediction using the model
prediction = best_model.predict(img_array)

# Get class labels
class_labels = ['Cat', 'Dog']

# Print prediction
predicted_class = class_labels[np.argmax(prediction)]
print("Predicted Class:", predicted_class)

# Display the image with predicted class
plt.imshow(img)
plt.title(f"True Label: {true_label}, Predicted Label: {predicted_class}")
plt.axis('off')
plt.show()
```

```
1/1 [==============================] - 0s 50ms/step
Predicted Class: Cat
```