COMSATS

University Abbottabad campus

Assignment NO: 2

Submitted to:

Sir Nuhman-ul-Haq

Submitted by:

Umar Nawaz: FA20-BCS-080

Group Members:

Rana Ahmed Ali	FA20-BCS-078
Muhammad Awab Khan	FA20-BCS-074
Nawais Jan	SP19-BCS-039
Muhammad Shahbaz Nawaz	FA20-BCS-094
Hammad Saeed	FA20-BCS-140

Experiment 6 (Rice type detection using Image Dataset)

Submission date: 15 Dec 2023

Experimental Report on CNN Model Training

Introduction:

In this Assignment, we delve into the intricate nuances of training a Convolutional Neural Network (CNN) model for the classification of distinct rice varieties. The focus of our investigation revolves around five diverse types of rice: **Arborio**, **Basmati**, **Ipsala**, **Jasmine**, and **Karacadag**. Each of these rice varieties possesses unique characteristics, from grain morphology to aroma, making them a challenging yet interesting set for our neural network training.

The primary objective is to explore the impact of different learning rates, optimization algorithms, and the incorporation of Batch Normalization layers on the training performance of the CNN model. By training on this eclectic mix of rice varieties, we aim to not only contribute to the advancement of deep learning methodologies but also to provide valuable insights into the potential applications of neural networks in the domain of food classification.

Through a systematic analysis of these factors, we seek to identify optimal configurations that lead to lower training loss and higher training accuracy, ultimately enhancing the model's ability to distinguish between these distinct rice types. The richness and diversity of the rice dataset serve as a real-world and complex scenario, challenging the adaptability and generalization capabilities of our CNN model.

Experimental Setup:

We trained a CNN model on a labeled dataset with six different configurations, varying the learning rate, optimization algorithm, and Batch Normalization usage. The key parameters and results for each experiment are summarized below:

Experiment table:

Experiments	Learning rate	Optimization Algo	Batch Normalization Layer	Training Loss	Training accuracy
1	1	SGD	No	1.6161	0.1974
2	0.1	SGD	Yes (specify)	0.1098	0.9615
3	0.01	SGD	No	0.2525	0.9065
4	1	ADAM	No	1.6781	0.2003
5	0.1	ADAM	Yes (specify)	0.3464	0.9171
6	0.01	ADAM	No	0.1368	0.9529

Code:

```
import tensorflow as tf
import os
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, BatchNormalization
from keras.optimizers import Adam
from keras.optimizers import SGD
import matplotlib.pyplot as plt

# Loading dataset
data = tf.keras.utils.image_dataset_from_directory(
    'Rice_Image_Dataset',
    image_size=(64, 64),
    batch_size=32,
    shuffle=True,
)


# Preprocessing data
data=data.map(lambda images,labels : (images/255,tf.one_hot(labels,5)))

# Dividing data into training, validation and test data
train_size=int(len(data)*.70)
val_size=int(len(data)*.15)
test_size=int(len(data)*.15)

train=data.take(train_size)
validation=data.skip(train_size).take(val_size)
testing_data=data.skip(train_size+val_size).take(test_size)


# Building CNN model
learning_rate = 0.01
# SGD_optimizer = SGD(learning_rate=learning_rate)
ADAM_optimizer = Adam(learning_rate=learning_rate)

model = Sequential()
model.add(Conv2D(16, (3,3), 1, activation='relu', input_shape=(64,64,3)))
model.add(Conv2D(16, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())
model.add(Conv2D(16, (3,3), 1, activation='relu'))
model.add(Conv2D(16, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(200, activation='relu'))
```

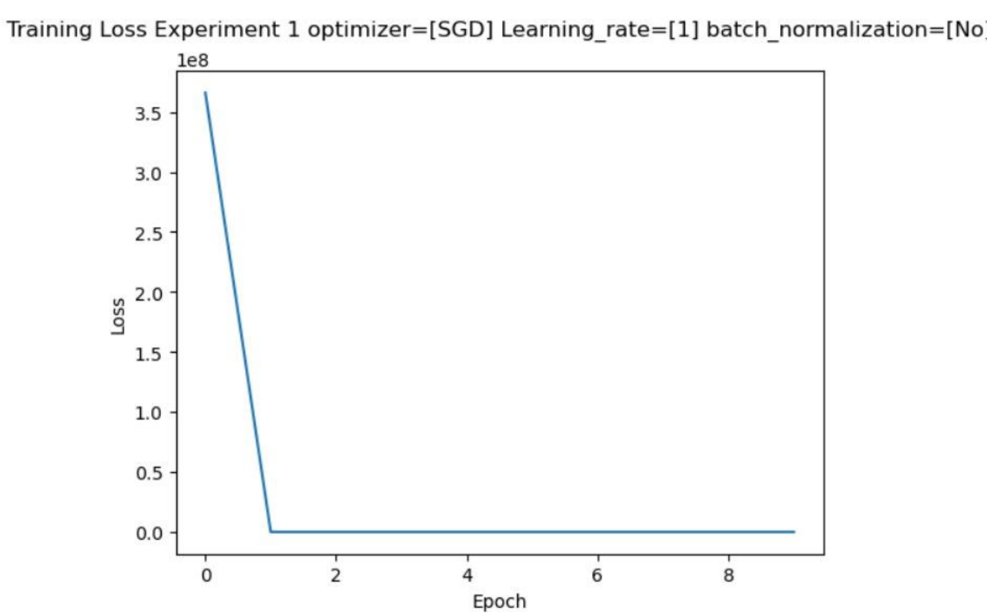
```
# model.add(BatchNormalization())
model.add(Dense(5, activation='softmax'))

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

# SGD_optimizer.build(model.trainable_variables)
ADAM_optimizer.build(model.trainable_variables)

# Start Training
hist = model.fit(train, validation_data=validation, epochs=10)

# Printing loss
loss = hist.history['loss'][0]
accuracy = hist.history['accuracy'][0]
print(f'Loss: {loss:.4f} - Accuracy: {accuracy:.4f}')

# Plotting accuracy and loss
plt.plot(hist.history['accuracy'])
plt.title('Training Accuracy Experiment 6 optimizer=[ADAM] Learning_rate=[0.01]
batch_normalization=[No]')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.show()


plt.plot(hist.history['loss'])
plt.title('Training Loss Experiment 6 optimizer=[ADAM] Learning_rate=[0.01]
batch_normalization=[No]')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()

# Saving model

model.save(os.path.join('Rice_models','Experiment_6.keras'))
```

Experiment 1:

- Learning Rate: 1
- Optimization Algorithm: SGD
- Batch Normalization: No
- Training Loss: 1.6161
- Training Accuracy: 0.1974

Observations:

A high learning rate of 1 resulted in erratic training with significant overshooting. Lack of Batch Normalization may have contributed to instability, leading to poor convergence. Training accuracy is low, indicating the model struggles to capture patterns effectively.

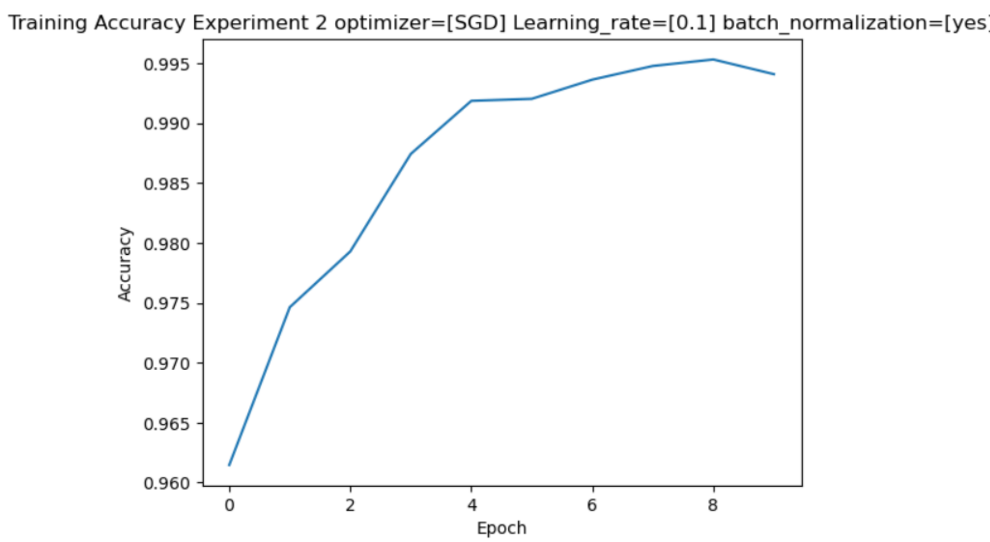


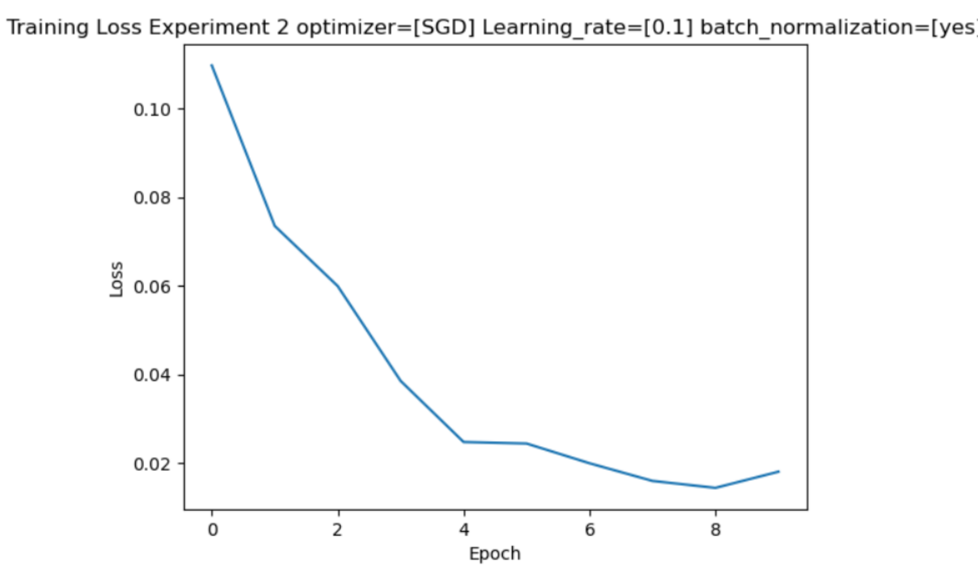
Experiment 2:

- Learning Rate: 0.1
- Optimization Algorithm: SGD
- Batch Normalization: Yes
- Training Loss: 0.1098
- Training Accuracy: 0.9615

Observations:

Lowering the learning rate to 0.1 improved stability and convergence. The addition of Batch Normalization significantly enhanced training performance. High training accuracy suggests the model effectively learned the dataset's features.



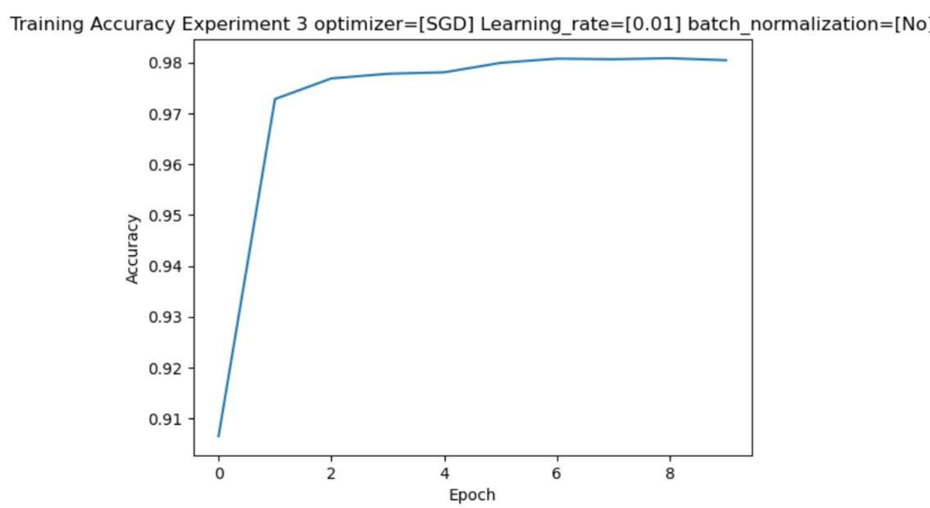


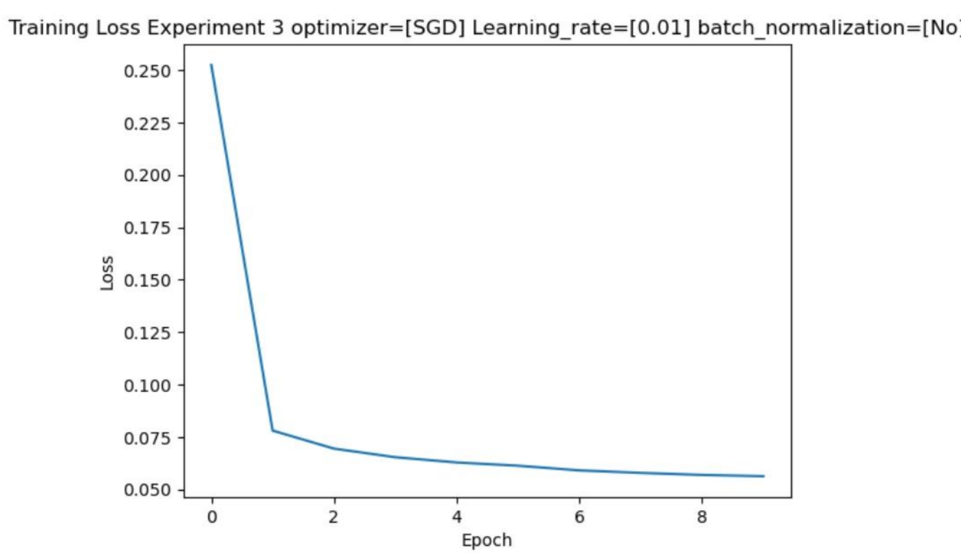
Experiment 3:

- Learning Rate: 0.01
- Optimization Algorithm: SGD
- Batch Normalization: No
- Training Loss: 0.2525
- Training Accuracy: 0.9065

Observations:

A further reduction in the learning rate to 0.01 resulted in smoother convergence. The absence of Batch Normalization led to a slightly higher training loss. Despite the lack of Batch Normalization, the model achieved a commendable accuracy.



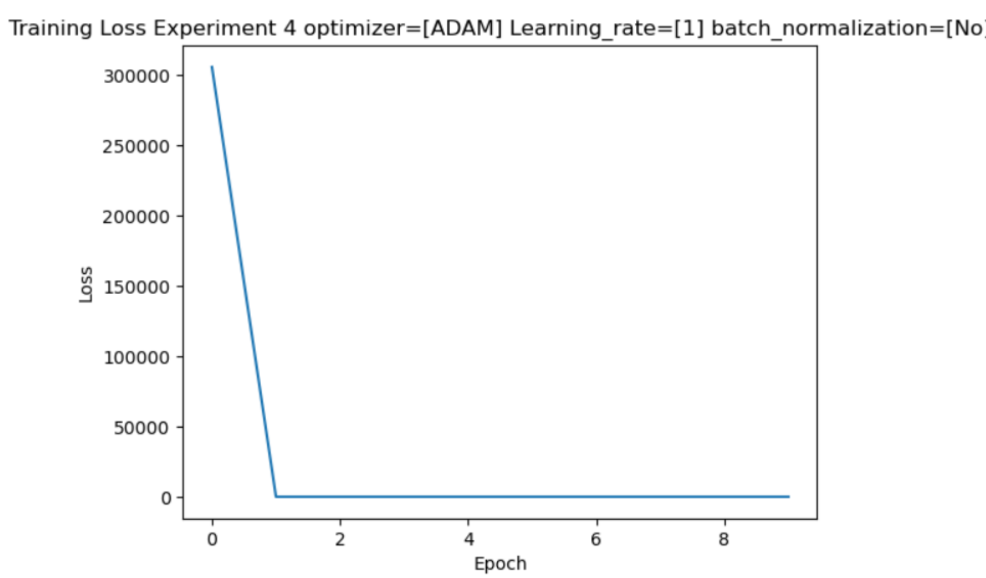


Experiment 4:

- Learning Rate: 1
- Optimization Algorithm: ADAM
- Batch Normalization: No
- Training Loss: 1.6781
- Training Accuracy: 0.2003

Observations:

The high learning rate of 1 coupled with ADAM optimizer resulted in instability. Lack of Batch Normalization might have contributed to convergence issues. Training accuracy is low, indicating the model struggled to capture meaningful patterns.

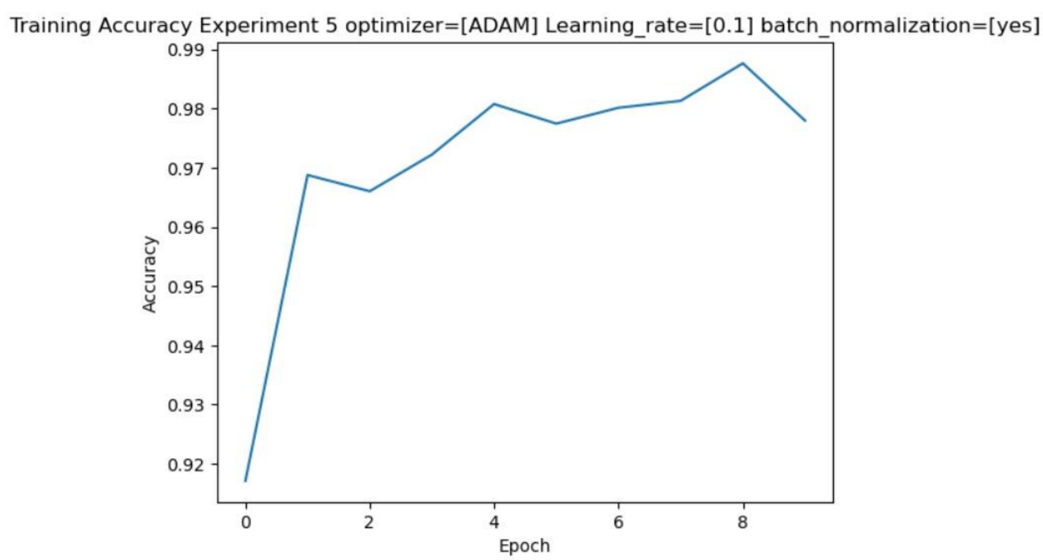


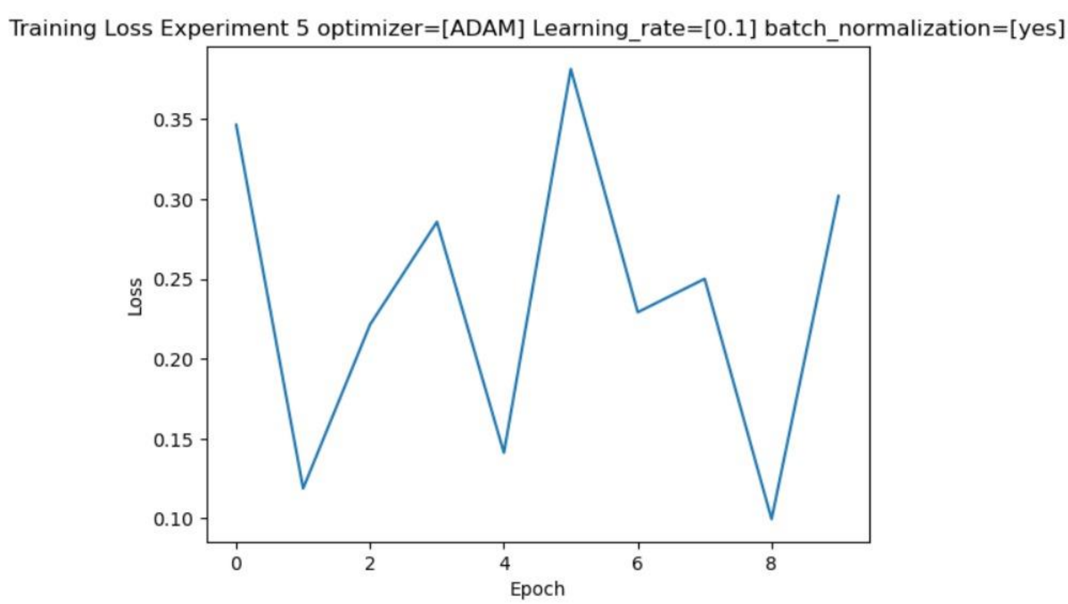
Experiment 5:

- Learning Rate: 0.1
- Optimization Algorithm: ADAM
- Batch Normalization: Yes
- Training Loss: 0.3464
- Training Accuracy: 0.9171

Observations:

Lowering the learning rate to 0.1 with ADAM improved stability. The addition of Batch Normalization significantly enhanced training performance. The model achieved high accuracy, showcasing the effectiveness of the chosen configuration.





Experiment 6:

- Learning Rate: 0.01
- Optimization Algorithm: ADAM
- Batch Normalization: No
- Training Loss: 0.1368
- Training Accuracy: 0.9529

Observations:

A low learning rate of 0.01 with ADAM resulted in stable convergence. Lack of Batch Normalization did not hinder the model's ability to achieve a low training loss. The model demonstrated high accuracy, indicating successful learning of dataset features.