Lab 5

Daniel Mehta

In [24]:
```
# imports
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
```

Load dataset and Preprocessing

In [25]:
```
# Load CIFAR 10 dataset
(x_train, y_train),(x_test, y_test) = cifar10.load_data()
```

In [26]:
```
# Only using 10,000 training images and 2000 testing images (full dataset was too large for my pc to handle)
x_train = x_train[:10000]
y_train = y_train[:10000]
x_test = x_test[:2000]
y_test = y_test[:2000]
```

In [27]:
```
# Flattening label arrays
y_train =y_train.flatten()
y_test =y_test.flatten()
```

In [28]:
```
#keeping 3 classes
target_classes = [0,3,5]
```

```
#boolean mask for filtering training and test sets
train_mask = np.isin(y_train, target_classes)
test_mask = np.isin(y_test, target_classes)

#filter images and labels
x_train = x_train[train_mask]
y_train = y_train[train_mask]
x_test = x_test[test_mask]
y_test = y_test[test_mask]

# remap labels to 0,1,2 so model output matches
class_map = {k: i for i, k in enumerate(target_classes)}
y_train = np.array([class_map[label] for label in y_train])
y_test = np.array([class_map[label] for label in y_test])
```

In [29]:
```
# normailze pixel values [0,1]
x_train =x_train.astype('float32')/255
x_test  =x_test.astype('float32')/255
```

In [30]:
```
# resize for ResNet50. to 224x224
x_train_resized = tf.image.resize(x_train, [224,224])
x_test_resized = tf.image.resize(x_test, [224,224])
```

In [31]:
```
# Printing to confirm the shapes
print("Training data shape:",x_train_resized.shape)
print("Training labels shape:", y_train.shape)
print("Test data shape:",x_test_resized.shape)
print("Test labels shape:", y_test.shape)
```

```
Training data shape: (2958, 224, 224, 3)
Training labels shape: (2958,)
Test data shape: (580, 224, 224, 3)
Test labels shape: (580,)
```

Data Augmentation and Train/Val Generators

In [32]:
```
# convert to numpy before splitting
x_train_resized =x_train_resized.numpy()
x_test_resized =x_test_resized.numpy()
```

In [33]:
```
# 80/20 split train/val
x_train_final, x_val, y_train_final, y_val = train_test_split(
    x_train_resized, y_train, test_size=0.2, random_state=5500
)
```

In [34]:
```
# createing a ImageDataGenerator for training with augmentation
train_datagen = ImageDataGenerator(
    rotation_range=10,
    horizontal_flip=True,
    zoom_range=0.05,
    brightness_range=[0.9, 1.1]
)
```

In [35]:
```
# doesnt change the validation images. keeps raw for accurate eval
val_datagen =ImageDataGenerator()
```

In [36]:
```
#create training and validation generators from numpy arrays
train_generator = train_datagen.flow(
    x_train_final, y_train_final,batch_size=32
)

val_generator = val_datagen.flow(
    x_val, y_val,batch_size=32
)
```

ResNet50 transfer learning model

In [37]:
```
# load base Resnet50
base_model = ResNet50(weights='imagenet', include_top=False,input_shape=(224, 224,3))
```

In [38]:
```
# freeze all base layer
for layer in base_model.layers:
    layer.trainable =False
```

```
# unfreeze last 20 layers
for layer in base_model.layers[-20:]:
    layer.trainable = True
```

In [39]:
```
# building full model
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(1024, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    Dense(3, activation='softmax')  #reduced CIFAR10 to 3 classes
])
```

Compile the model

In [40]:
```
# compiling using Adam optimizaer
model.compile(
    optimizer=Adam(), # adaptive learning rate
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

Adding Early Stopping

In [41]:
```
# defining EarlyStopping to stop training when the val loss stops improving
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=3, #min of 3 epochs before stopping
    restore_best_weights=True # goes back to best model weight
)
```

Training the model

In [42]:
```
history = model.fit(
    train_generator,
```

```
    epochs=500,
    validation_data=val_generator,
    callbacks=[early_stop], # Early stopping
    verbose=1
)
```

```
Epoch 1/500
74/74 [==============================] - 42s 525ms/step - loss: 1.2151 - accuracy: 0.3309 - val_loss: 5.8406 - val_ac
curacy: 0.3209
Epoch 2/500
74/74 [==============================] - 38s 517ms/step - loss: 1.1708 - accuracy: 0.3369 - val_loss: 4.8322 - val_ac
curacy: 0.3649
Epoch 3/500
74/74 [==============================] - 38s 518ms/step - loss: 1.1357 - accuracy: 0.3238 - val_loss: 6.8729 - val_ac
curacy: 0.3142
Epoch 4/500
74/74 [==============================] - 38s 518ms/step - loss: 1.1550 - accuracy: 0.3478 - val_loss: 6.2853 - val_ac
curacy: 0.3176
Epoch 5/500
74/74 [==============================] - 38s 519ms/step - loss: 1.1784 - accuracy: 0.3314 - val_loss: 19.8649 - val_a
ccuracy: 0.3209
```

Plotting and Evaluation

In [43]:
```
# Converting accuracy values to percentages
trainAcc = [100*x for x in history.history['accuracy']]
testAcc = [100*x for x in history.history['val_accuracy']]
```

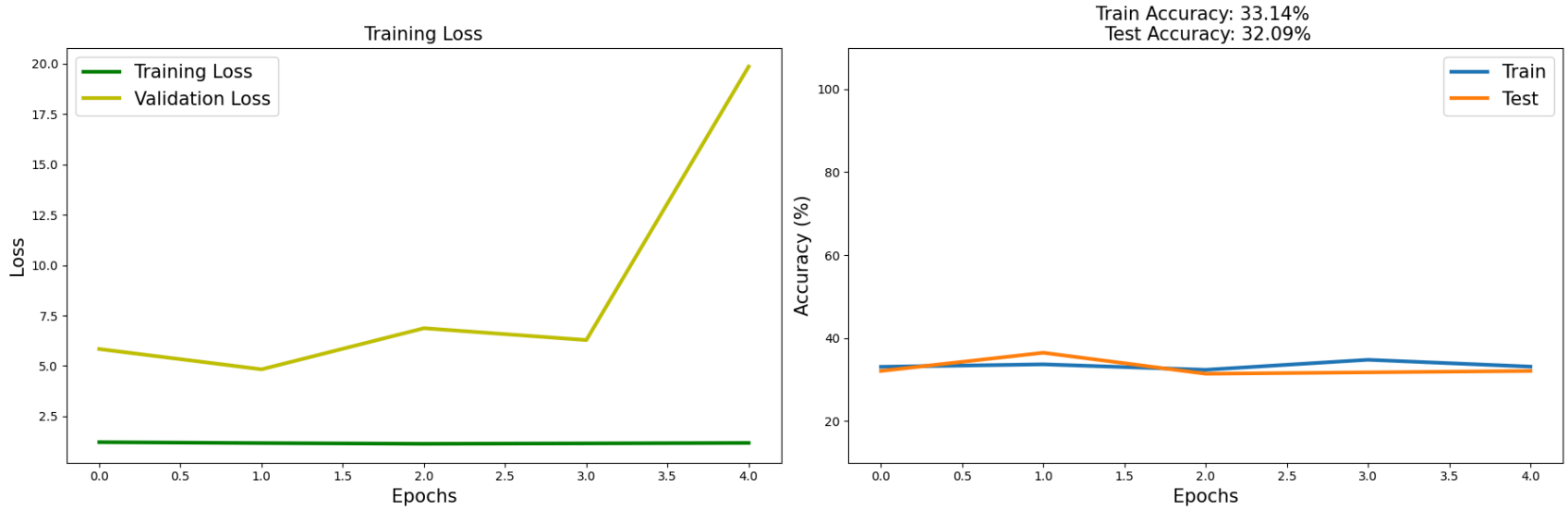
In [44]:
```
fig, ax = plt.subplots(1, 2, figsize=(18, 6))

# loss plot
ax[0].plot(history.history['loss'], 'g', lw=3, label='Training Loss')
ax[0].plot(history.history['val_loss'], 'y', lw=3, label='Validation Loss')
ax[0].set_xlabel('Epochs', fontsize = 15)
ax[0].set_ylabel('Loss', fontsize = 15)
ax[0].legend(fontsize = 15)
ax[0].set_title('Training Loss', fontsize = 15)

# accuracy plot
ax[1].plot(trainAcc, label ='Train', lw = 3)
ax[1].plot(testAcc, label ='Test', lw = 3)
```

```
ax[1].set_xlabel('Epochs', fontsize = 15)
ax[1].set_ylabel('Accuracy (%)', fontsize = 15)
ax[1].set_ylim([10,110])
ax[1].set_title(f'Train Accuracy: {trainAcc[-1]:.2f}% \n Test Accuracy: {testAcc[-1]:.2f}%', fontsize = 15)
ax[1].legend(fontsize = 15)

plt.tight_layout()
plt.show()
```



In [45]:
```
val_loss_final, val_acc_final = model.evaluate(val_generator, verbose=0)
print(f"Final Validation Accuracy: {val_acc_final:.4f}")
print(f"Final Validation Loss: {val_loss_final:.4f}")
```

```
Final Validation Accuracy: 0.3649
Final Validation Loss: 4.8322
```

In [46]:
```
print("Train Accuracy History:", trainAcc)
print("Test Accuracy History:", testAcc)
```

```
Train Accuracy History: [33.093827962875366, 33.68554413318634, 32.37531781196594, 34.78444516658783, 33.136093616485
596]
Test Accuracy History: [32.094594836235046, 36.4864856004715, 31.418919563293457, 31.756755709648132, 32.094594836235
046]
```

In [ ]: