Deniz Solmaz
Eğitmenin Adı
29 Aralık 2023

ASSIGNMENT 4

Image Classification with Convolutional Neural Networks

Certainly, here's a more detailed version of the explanation for your Pages document:

1. Finding Image Paths (`find_image_paths_with_n_images` function): In the find_image_paths_with_n_images function, the decision was made to focus on 15 classes, each with at least 300 images, to ensure efficient training and manageable epoch times. This approach balances the need for a diverse dataset with computational efficiency. For each of these classes, the images are divided into 200 for training, 50 for validation, and 50 for testing, which allows for a comprehensive evaluation of the model's performance across different data sets. This strategy is a practical compromise to handle large datasets effectively while ensuring sufficient data variation for robust model training and evaluation.

2. Data Segregation: In this stage, images from each class are systematically divided into three distinct datasets: training, validation, and testing. Specifically, for every class, the first 200 images are allocated for training, the subsequent 50 images for validation, and the final 50 images for testing. This methodical separation is crucial for effectively training the CNN model and accurately evaluating its performance across different data samples, ensuring robustness and reliability in the model's predictive capabilities.

3. ImageDataGenerators: These generators are instrumental in the efficient loading and processing of images. They perform crucial tasks such as rescaling (normalizing) the images and organizing them into batches suitable for the CNN. The 'flow_from_dataframe' method is a key feature here, as it ensures that these generators are seamlessly linked with the prepared pandas dataframes, guaranteeing that each image is correctly associated with its

label. This step is essential for maintaining the integrity and accuracy of the model training process.

4. CNN Model Architecture (`build_model` function): This function is responsible for constructing the sequential CNN model. The model is composed of several convolutional layers, each followed by batch normalization and max-pooling layers. These layers are foundational in CNNs for effective feature extraction and crucial in reducing the risk of overfitting. The model concludes with fully connected dense layers that perform the classification task. Importantly, a dropout layer is included after the dense layer. This dropout layer is vital as it randomly ignores a portion of the neurons (set at 50% in this case) during training, which significantly helps in preventing the model from overfitting and ensures that it learns generalized patterns that are not overly reliant on the training data. This structure makes the model adept at handling complex image classification tasks, enhancing its performance and accuracy.

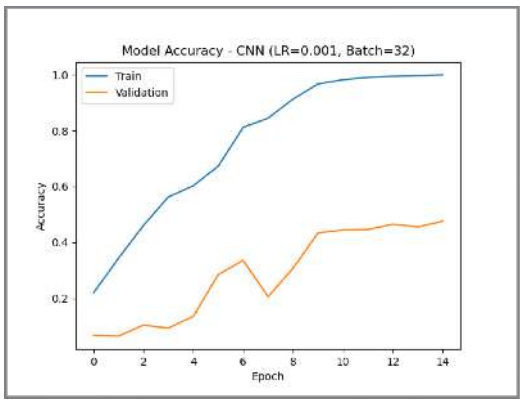
Drop-Out:

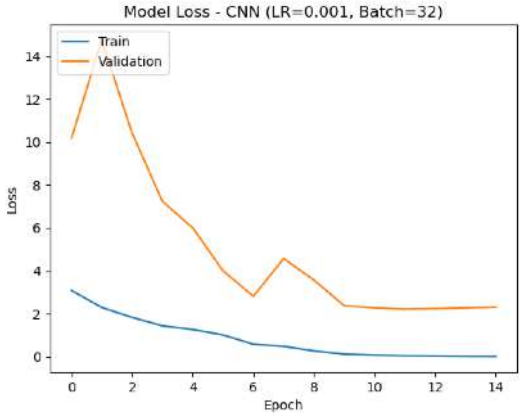
In the provided CNN model, dropout is added after the dense layer with 1024 neurons. This placement is strategic. Dense layers, especially ones with a large number of neurons, are prone to overfitting due to their capacity to learn complex patterns in the data. By introducing dropout here, a fraction of the neurons (50% in this case) are randomly ignored during each training iteration. This prevents the model from becoming too reliant on any specific set of neurons, thus reducing overfitting and encouraging the network to learn more robust features that generalize better to unseen data.

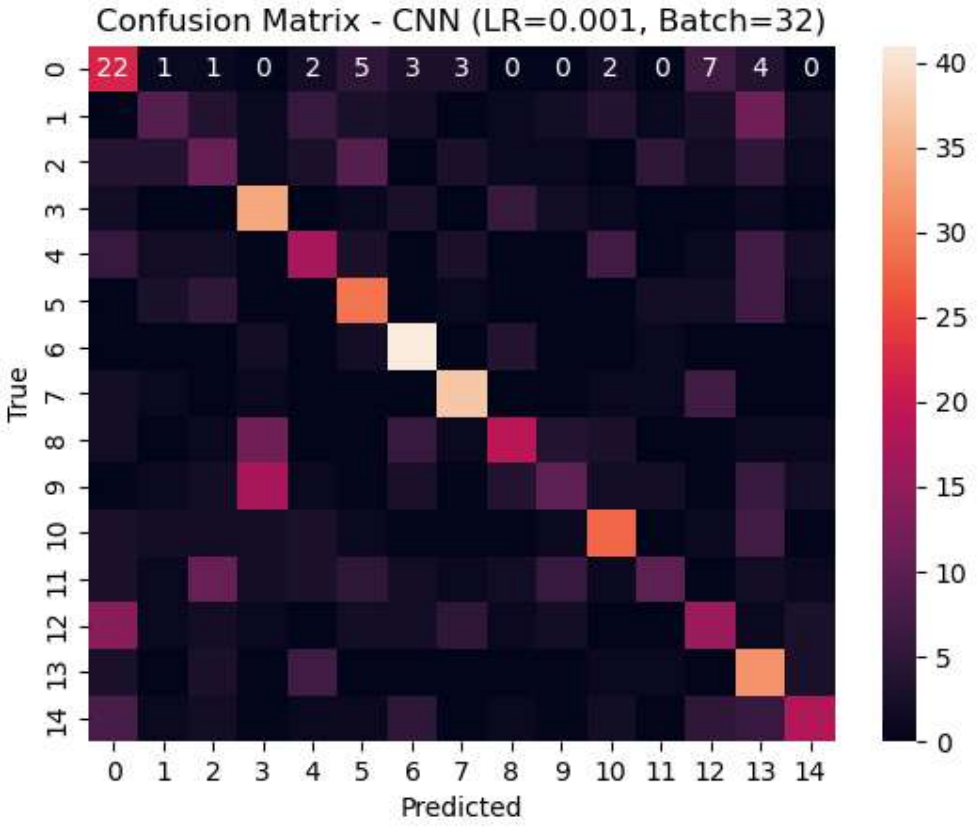
DIFFER IN LEARNING RATE AND BATCH SIZE

DIFFER IN LEARNING RATE AND BATCH SIZE

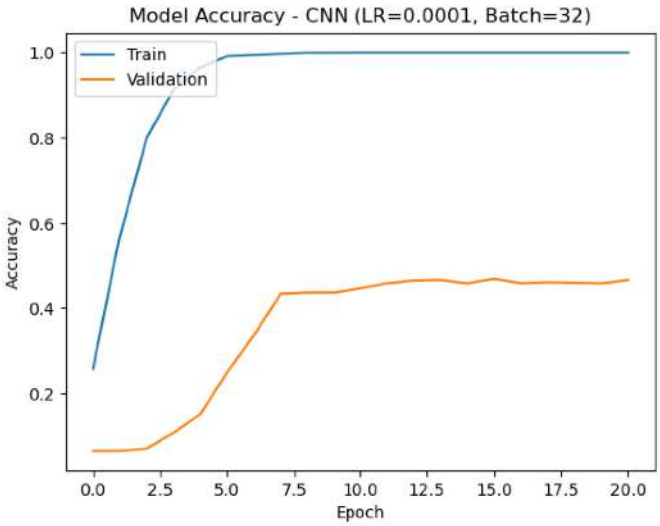
CNN LEARNING RATE 0,001 BATCH SIZE 32

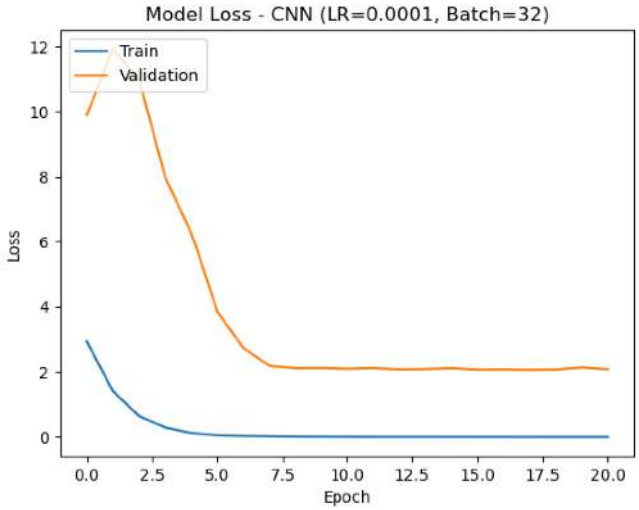


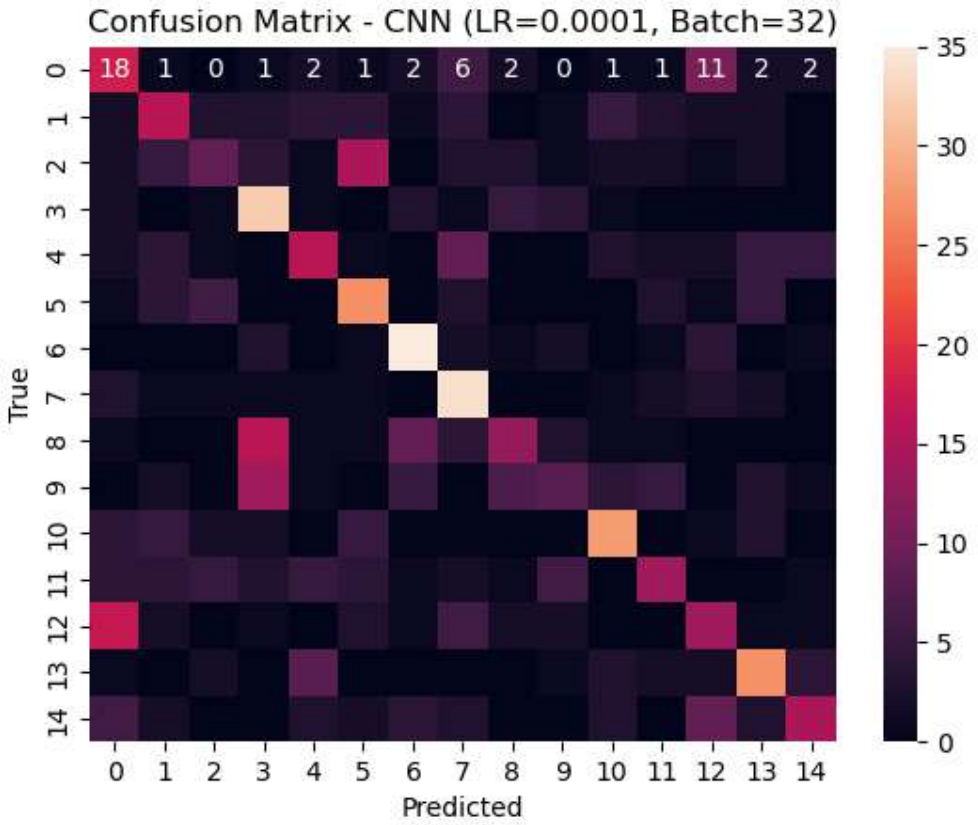




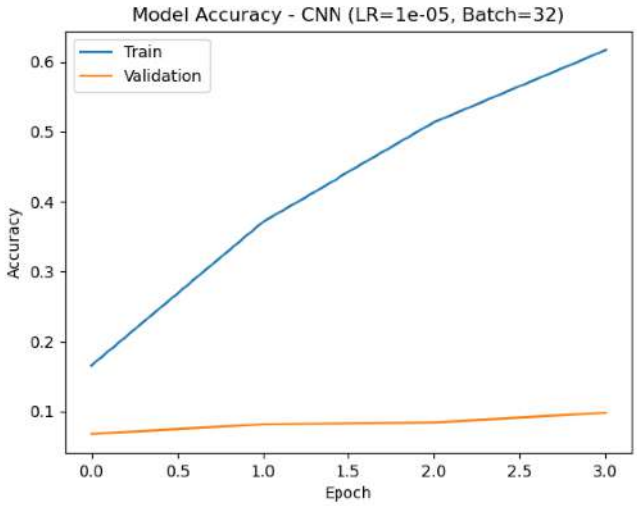
CNN LEARNING RATE 0,0001 BATCH SIZE 32

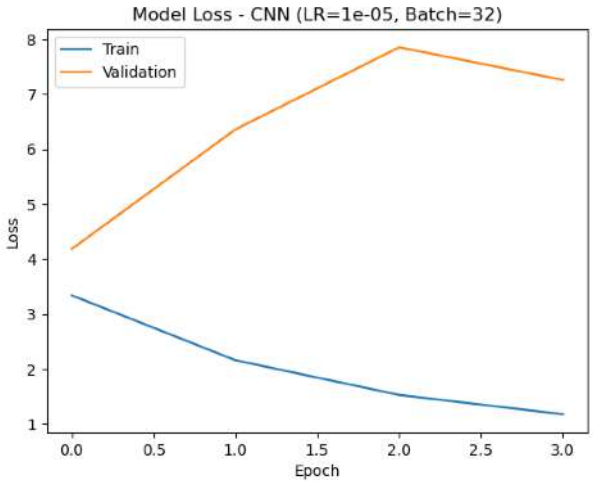


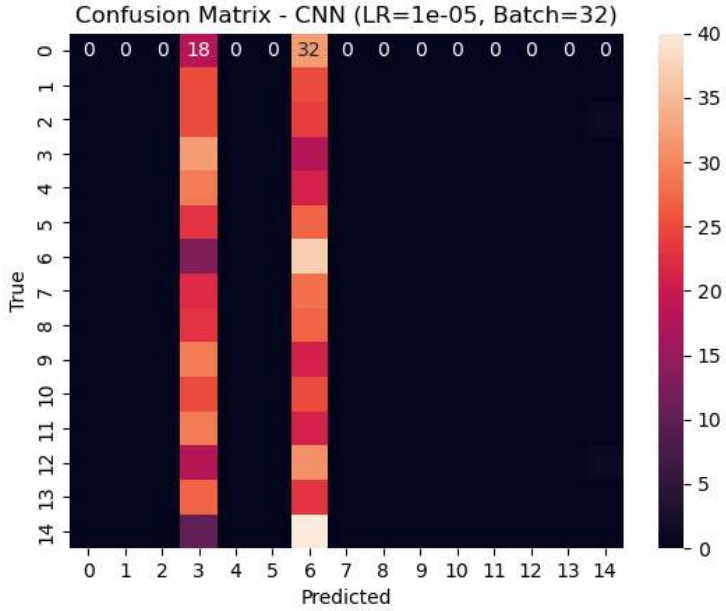




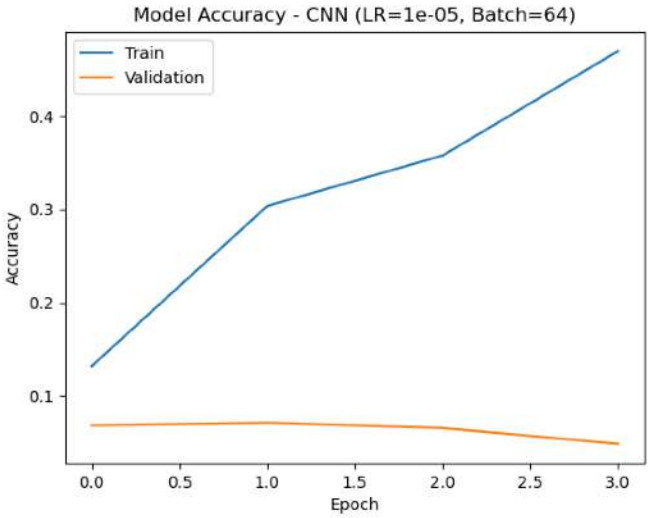
CNN LEARNING RATE 0,00001 BATCH SIZE 32

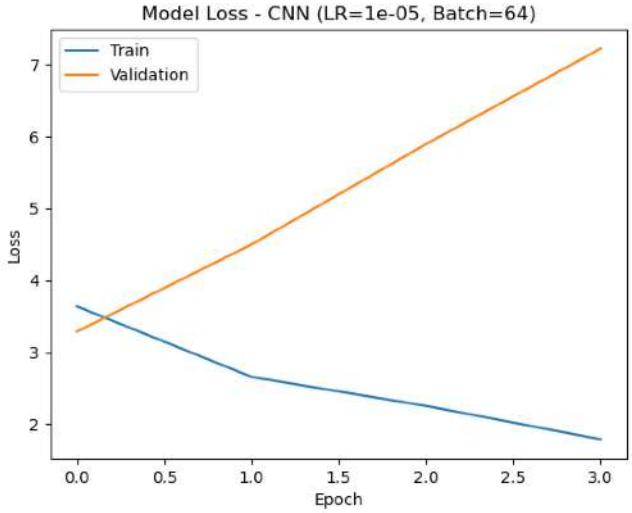


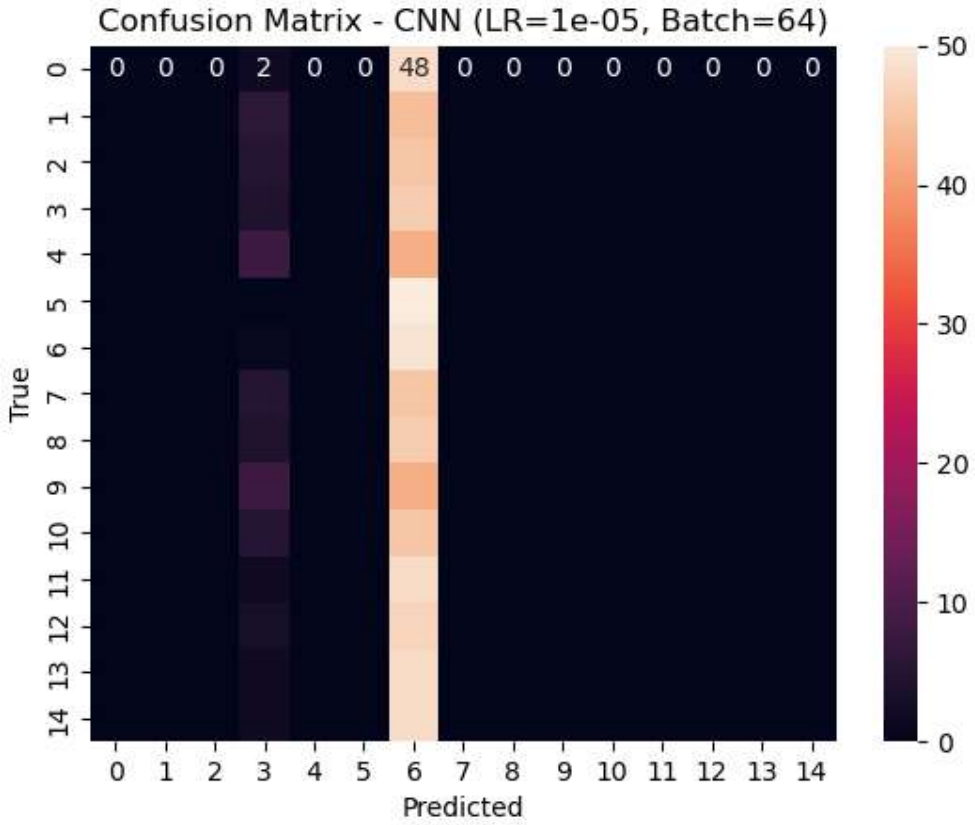




CNN LEARNING RATE 0,00001 BATCH SIZE 64







SAME-LEARNING RATE AND DIFFERENT BATCH SIZE

The results from the two sets of training runs with different batch sizes, while keeping the learning rate constant, are observed. In the first set with a batch size of 64, it was noted that the training accuracy increased gradually, whereas the validation accuracy remained relatively flat. This discrepancy suggests that the model might be overfitting to the training data. In terms of loss, the training loss decreased significantly, but the validation loss increased, reinforcing the indication of overfitting. In the second set with a batch size of 32, the training accuracy improved markedly, surpassing the results from the larger batch size. The validation accuracy, however, showed only a slight improvement, still displaying signs of overfitting. The training loss decreased consistently, while the validation loss showed some volatility, peaking towards the end of the epochs. These observations suggest that reducing the batch size helped the model to learn better, as indicated by the higher training accuracy, but the model's generalization to the validation set did not improve correspondingly. It may be beneficial to explore additional regularization techniques or hyperparameter adjustments to mitigate overfitting and enhance validation performance.

SAME BATCH SIZE DIFFERENT LEARNING RATE

In the experiments conducted, it was observed that when a higher learning rate of 0.001 was used, the training accuracy improved significantly over epochs, yet the validation accuracy fluctuated and remained much lower than the training accuracy, indicating a possible overfitting. The loss graphs confirmed this, showing a decreasing trend in training loss and an unstable pattern in validation loss. On the other hand, when the learning rate was decreased to 0.0001, the training accuracy increased at a slower rate, achieving a high level of accuracy by the end of the epochs. However, the validation accuracy plateaued early, suggesting that the model was not generalizing well beyond the training data. The loss for training decreased smoothly, and the validation loss decreased and plateaued, which could indicate the need for more epochs or adjustments in learning rate to improve model generalization.

DROP-OUT ANALYSIS:

Dropout Rate: 0.2
Validation Loss: 2.1896321773529053, Validation Accuracy: 0.4426666796207428
Test Loss: 2.4332220554351807, Test Accuracy: 0.40933331847190857

Dropout Rate: 0.3
Validation Loss: 2.182781934738159, Validation Accuracy: 0.4440000057220459
Test Loss: 2.4925696849823, Test Accuracy: 0.41733333468437195

Dropout Rate: 0.4
Validation Loss: 2.1298892498016357, Validation Accuracy: 0.4466666579246521
Test Loss: 2.338026285171509, Test Accuracy: 0.4000000059604645

Dropout Rate: 0.5
Validation Loss: 2.1176722049713135, Validation Accuracy: 0.4426666796207428
Test Loss: 2.4004738330841064, Test Accuracy: 0.41333332657814026

An analysis of dropout rates in a CNN model has been performed, and it was observed that increasing the dropout rate from 0.2 to 0.5 did not significantly improve validation accuracy, which remained around 44%. However, the test accuracy peaked at a dropout rate of 0.3 with approximately 41.7% and showed a slight decrease as the dropout rate increased further. It was noted that the lowest validation and test losses were achieved with a dropout rate of 0.4 and 0.5, respectively. This suggests that a higher dropout rate may help in reducing overfitting to some extent, but does not necessarily lead to better accuracy on this dataset.

PART 2

two models based on the VGG16 architecture, pretrained on ImageNet, are prepared for a classification task. The first model is characterized by all layers being set as non-trainable, which means that the learned weights from ImageNet will remain unchanged during training. This technique is typically employed when the dataset size is small to avoid overfitting and to utilize the model's existing feature extraction capabilities. Conversely, in the second model, the last eight layers are set as trainable. This allows for the adaptation of the model's more abstract features to the specifics of the new dataset, which can be beneficial when the available data is sufficient to support such fine-tuning without leading to overfitting. In both cases, the models are extended by adding a new top section of fully connected layers, concluding with a softmax layer for classification. These modifications are aimed at tailoring the pre-trained network to perform on a specific task involving a different number of classes than the original ImageNet challenge.

EARLY-STOPPING

Early stopping was utilized in the training process, which is a method where training is ceased when the model's performance on the validation set no longer improves for a set number of epochs, in this case, indicated by the 'patience' parameter. The advantage of early stopping is that it reduces the risk of overfitting by preventing the model from learning idiosyncrasies in the training data that do not generalize to new data. However, one disadvantage is that it may stop the training before the model has fully converged, potentially leading to a suboptimal set of weights if the 'patience' parameter is not appropriately set. It requires careful selection of the patience level to balance between sufficient training and overfitting.

FINE-TUNING

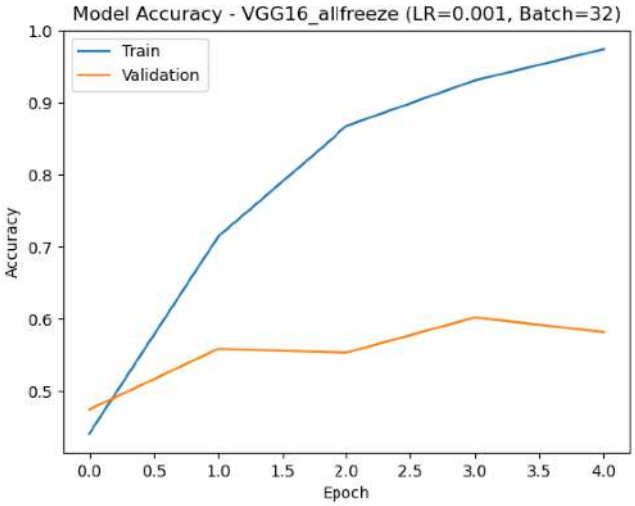
Fine-tuning is the process whereby the deeper layers of a pre-trained neural network are adjusted to make them more relevant for a specific task. It is often conducted after the top layers of the network have been re-trained or replaced to cater to the new task. The practice of freezing the rest of the network while training only the fully connected (FC) layers is due to the fact that the initial layers capture universal features like edges and textures that are applicable to a wide variety of tasks, while the later layers become more task-specific. Fine-tuning the last layers allows the model to adapt these specific features to the new task without the need for extensive retraining from scratch.

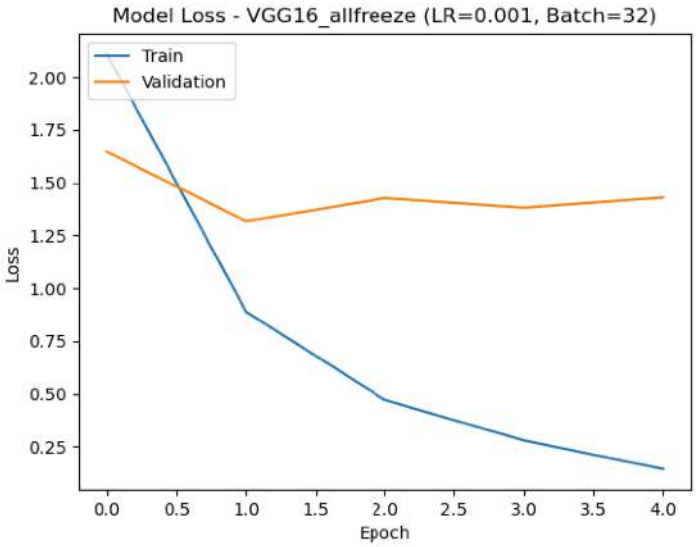
```python
# Load the VGG16 model, pretrained on ImageNet data, excluding the top (fully connected) layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(128, 128, 3))

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

# Create a new top (fully connected) layers for your specific classification task
x = base_model.output
x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)
model_allfreeze = Model(inputs=base_model.input, outputs=predictions)
```

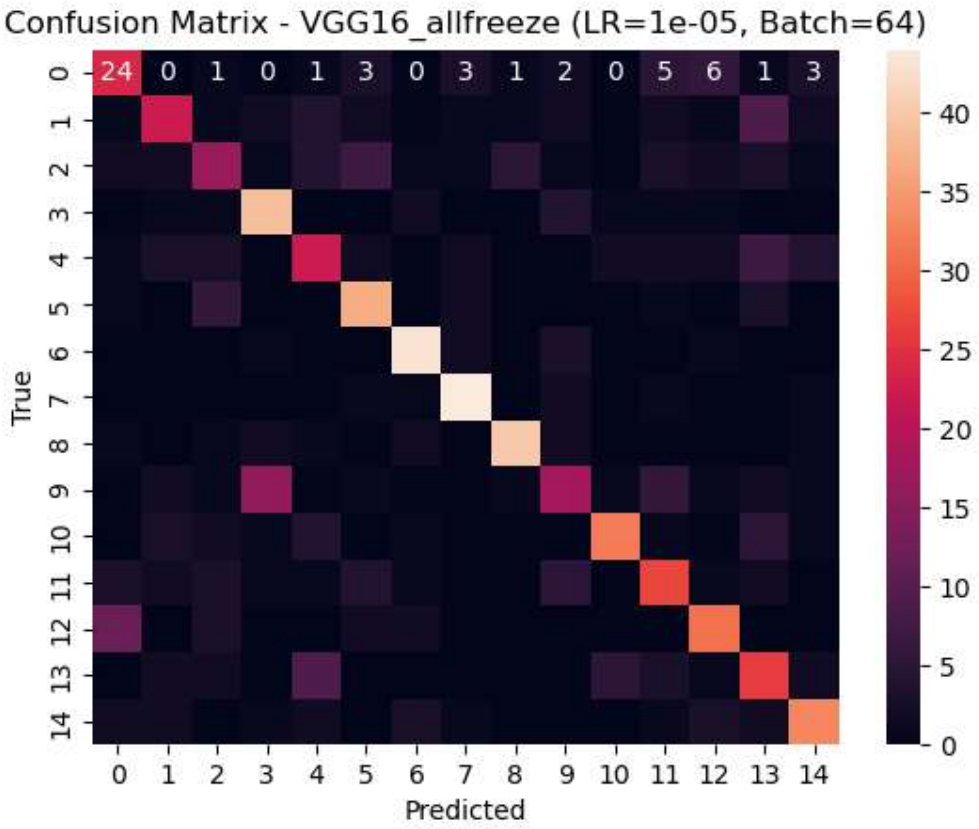
```python
# Load the VGG16 model, pretrained on ImageNet data, excluding the top (fully connected) layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(128, 128, 3))

# Freeze all layers of the base model
for layer in base_model.layers[:-8]:  # Freezes all layers except the last two convolutional blocks
    layer.trainable = False
for layer in base_model.layers[-8:]:
    layer.trainable = True

# Create a new top (fully connected) layers for your specific classification task
x = base_model.output
x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)
model_first2 = Model(inputs=base_model.input, outputs=predictions)
```

FULL-FREEZE ANALYSIS

LR=0.001 BATCH SIZE 32

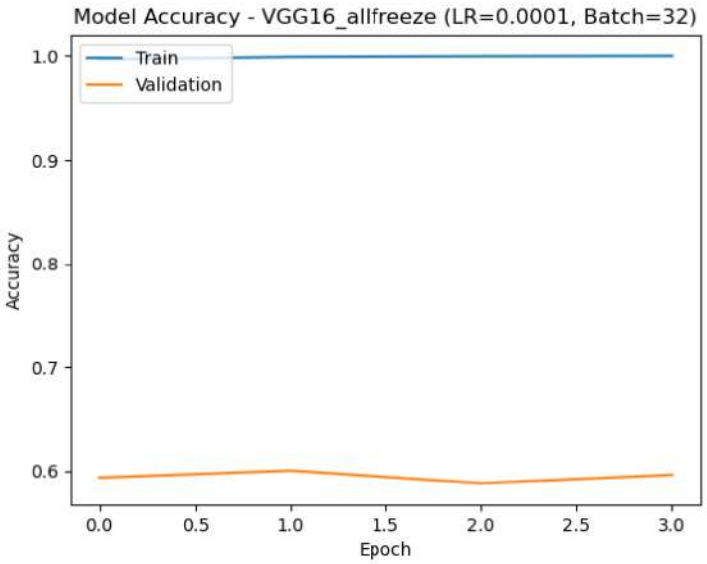


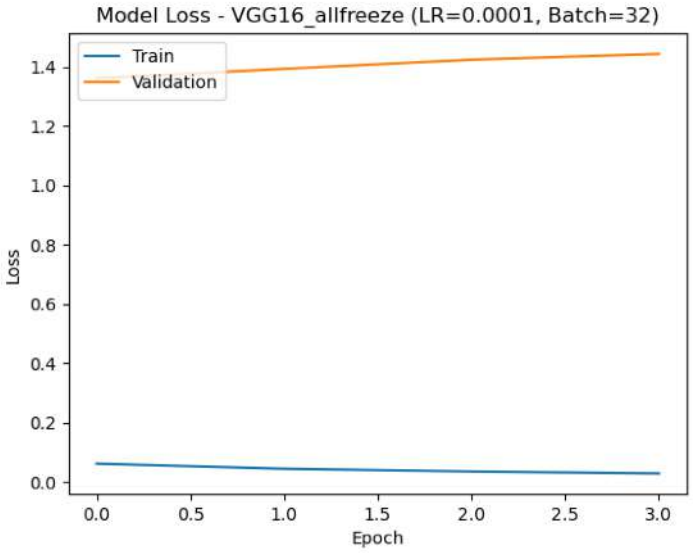


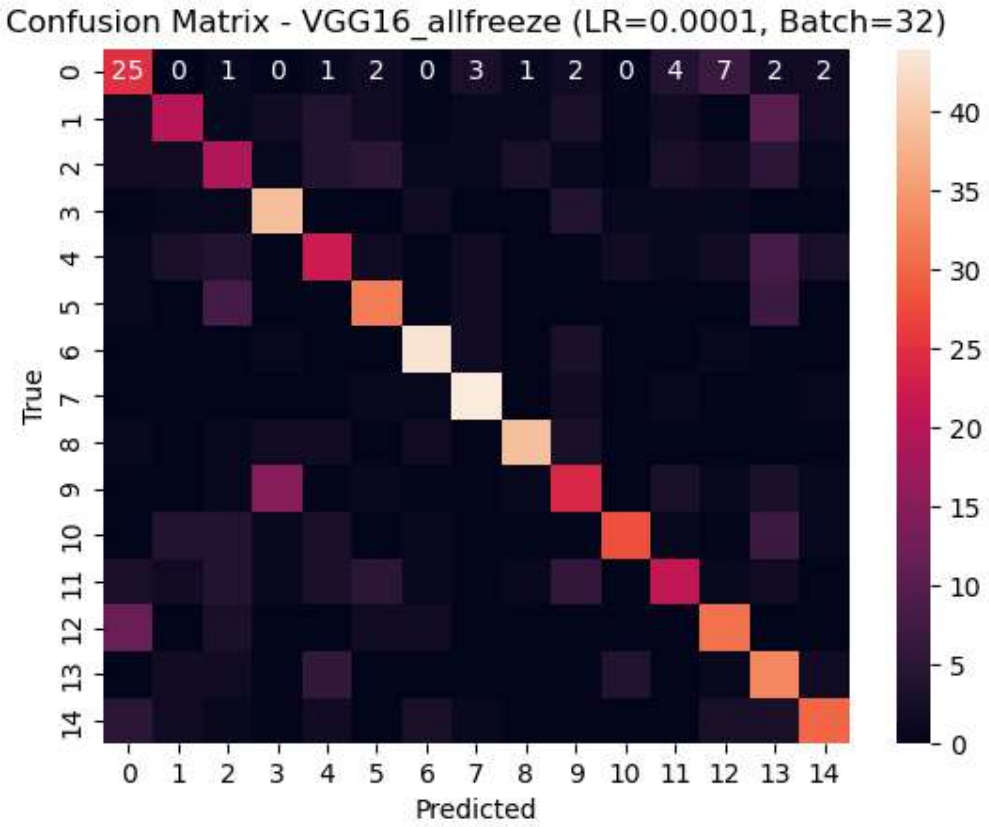


FULL-FREEZE ANALYSIS

LR=0.0001 BATCH SIZE 32

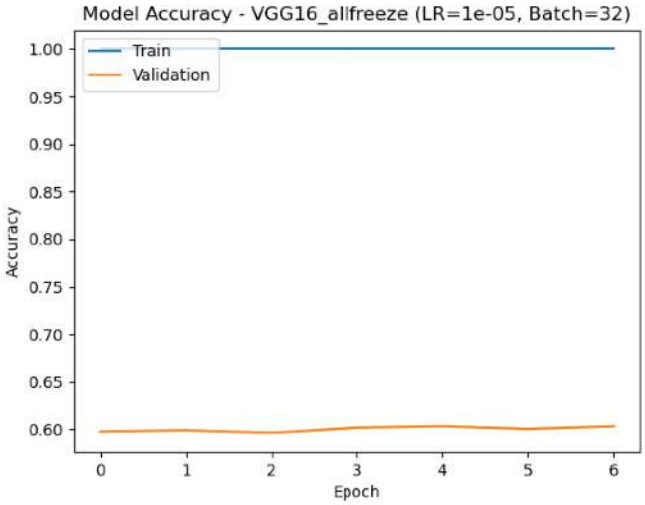


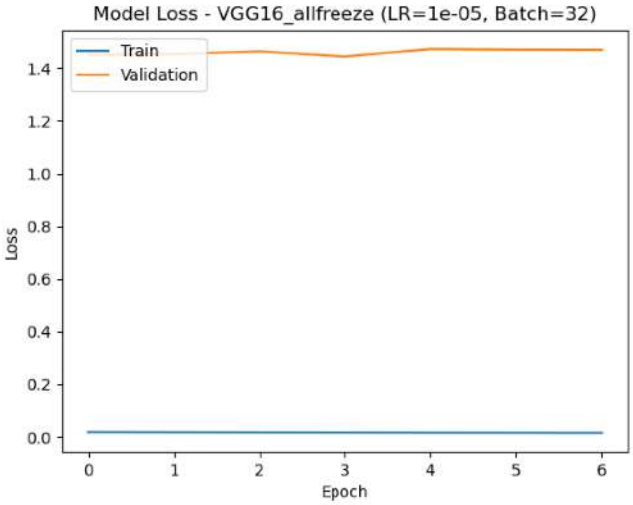


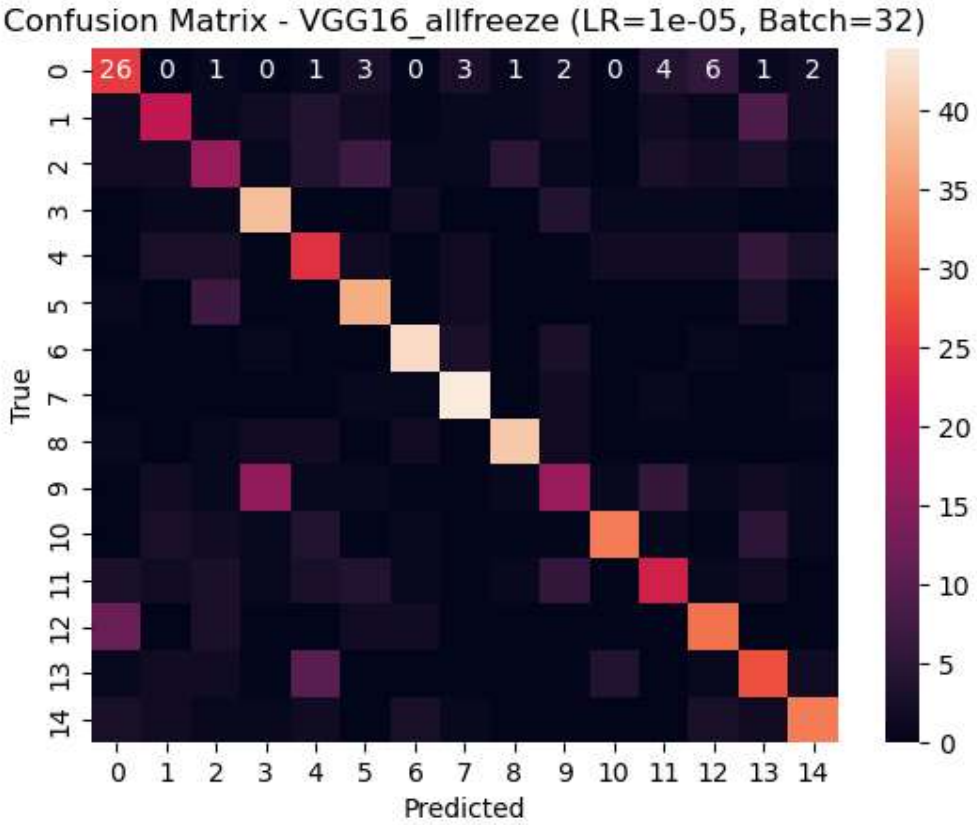


FULL-FREEZE ANALYSIS

LR=0.00001 BATCH SIZE 32

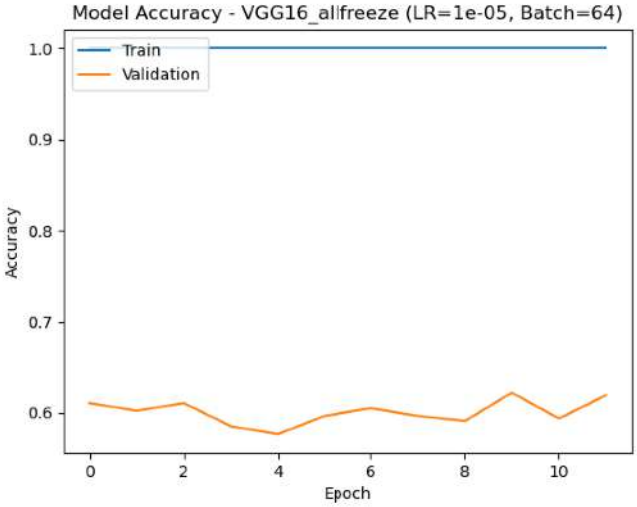


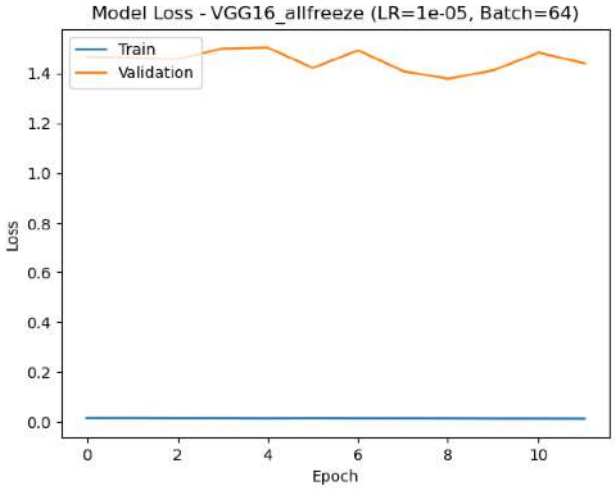


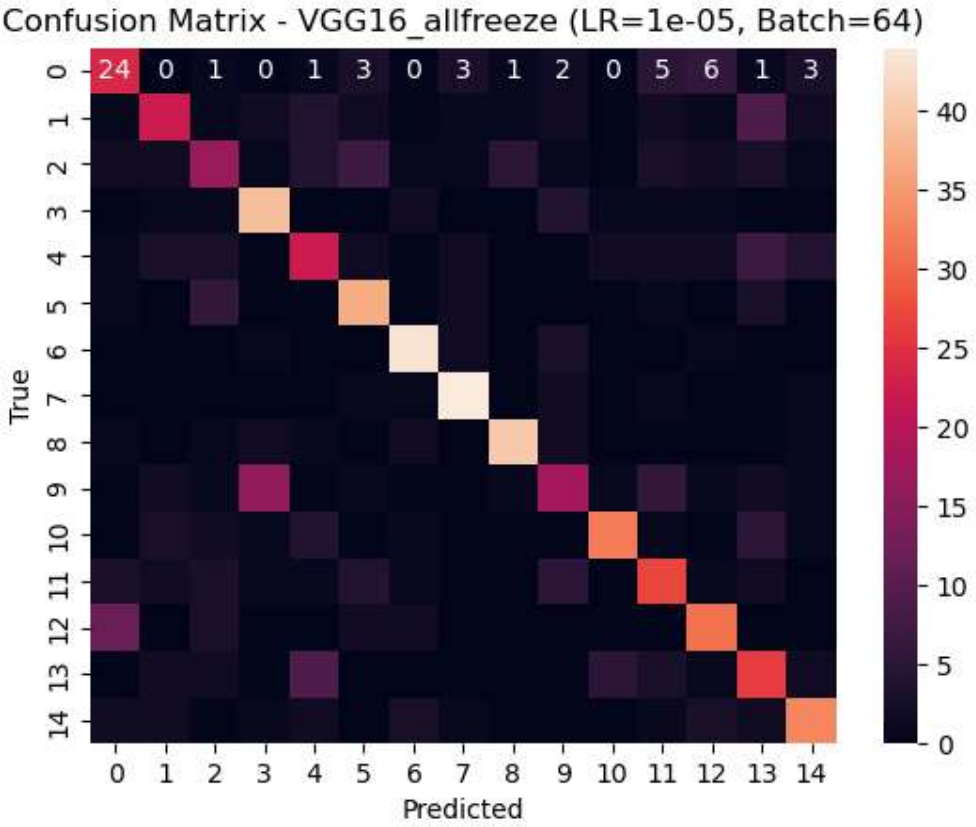


FULL-FREEZE ANALYSIS

LR=0.00001 BATCH SIZE 64







COMMENTING ABOUT

FULL-FREEZE RESULTS

In the VGG16 model training graphs with different learning rates, a consistent pattern was observed. Training accuracy was significantly higher than validation accuracy across all learning rates, indicating potential overfitting. Notably, with lower learning rates (1e-05), the gap between training and validation accuracy was less pronounced compared to higher learning rates (0.001), suggesting better generalization at lower learning rates. However, the validation accuracy did not improve substantially, which might suggest that further tuning or a different approach might be necessary to enhance model performance on unseen data.

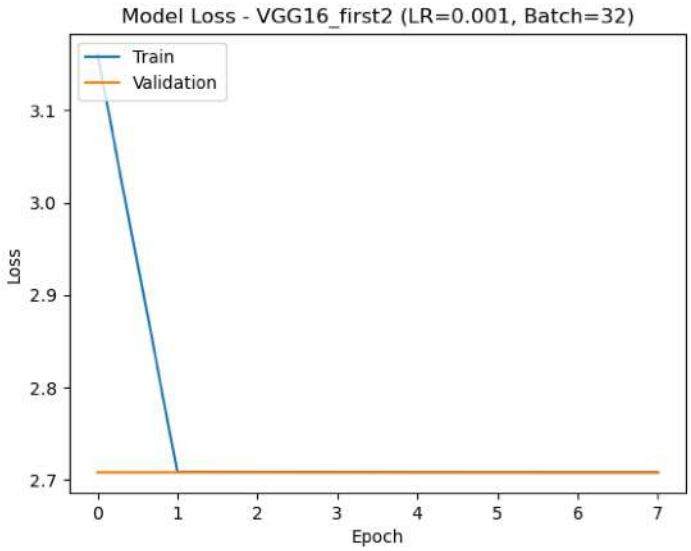
FULL-FREEZE VGG AND CNN RESULT (Lr=0.001 batch size=32)

The VGG16 model, with all layers frozen except the fully connected (FC) layers, displays a high training accuracy that significantly surpasses the validation accuracy, suggesting overfitting. Similarly, the custom CNN model exhibits overfitting, as indicated by its training accuracy surpassing the validation accuracy by a substantial margin. For both models, the training loss decreases smoothly, yet the validation loss either plateaus or increases, further implying that the models are not generalizing well to unseen data.

LAST 2 LAYER UNFREEZE VGG16

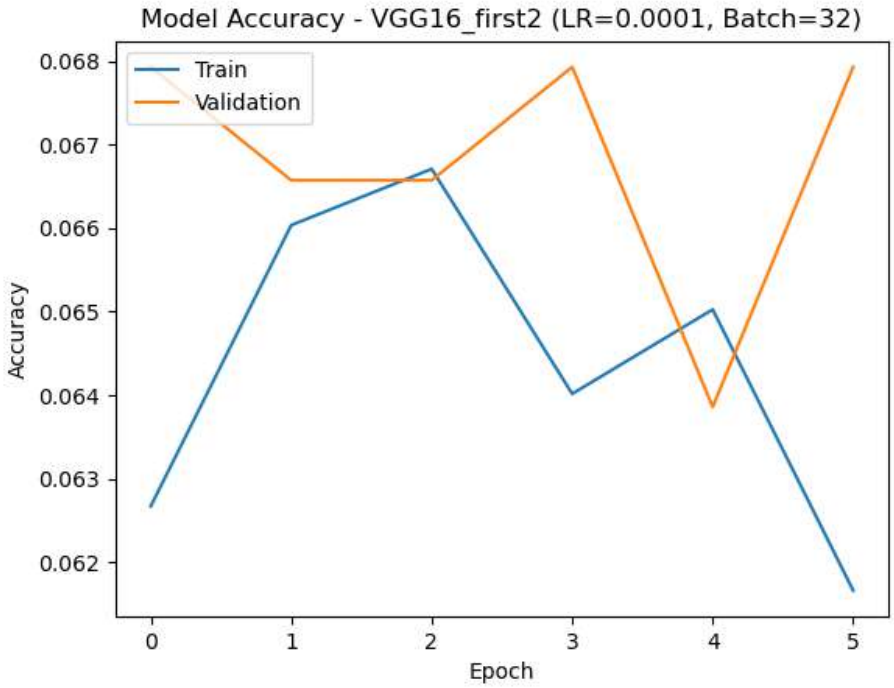
LAST 2-UNFREEZE VGG16

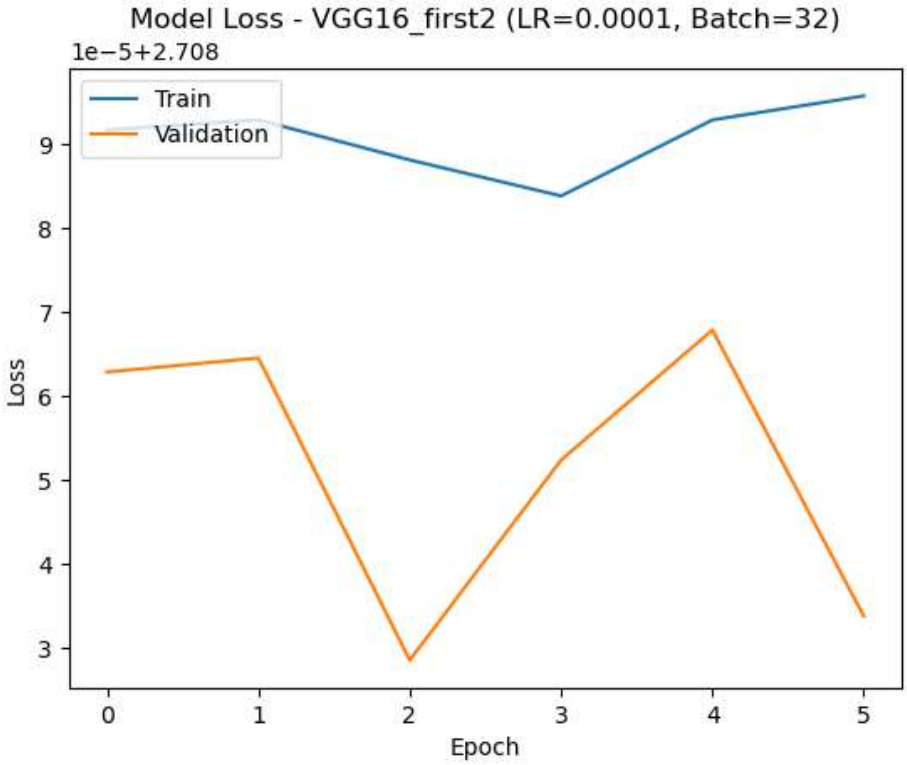
LR=0.001 BATCH SIZE 32



LAST 2-UNFREEZE VGG16

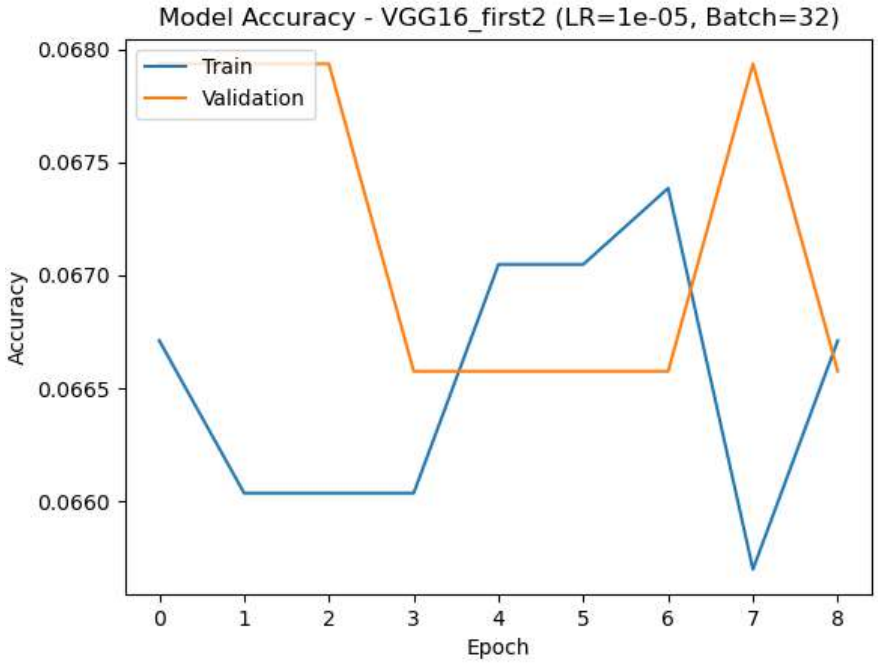
LR=0.0001 BATCH SIZE 32

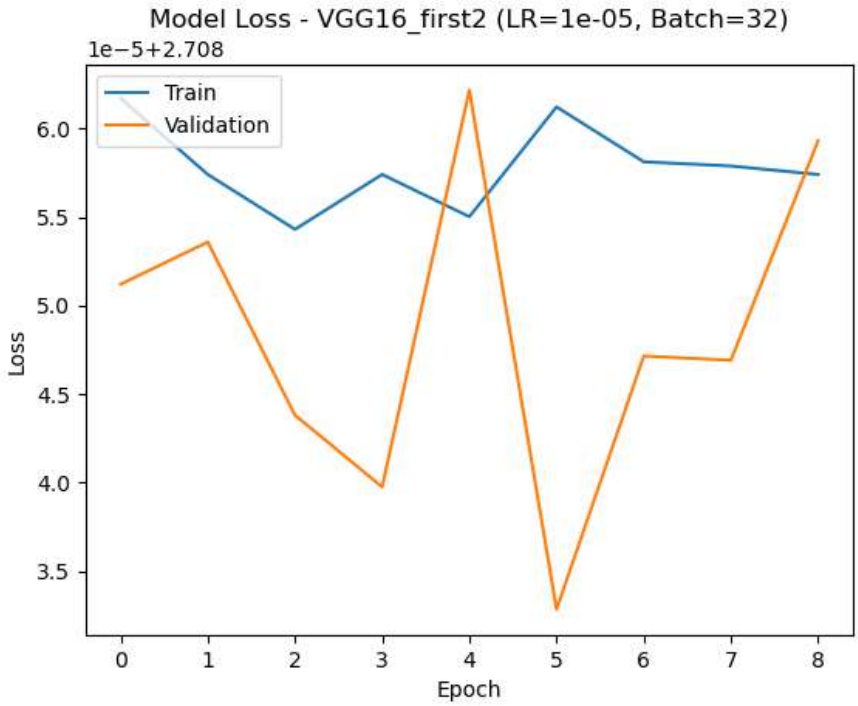




LAST 2-UNFREEZE VGG16

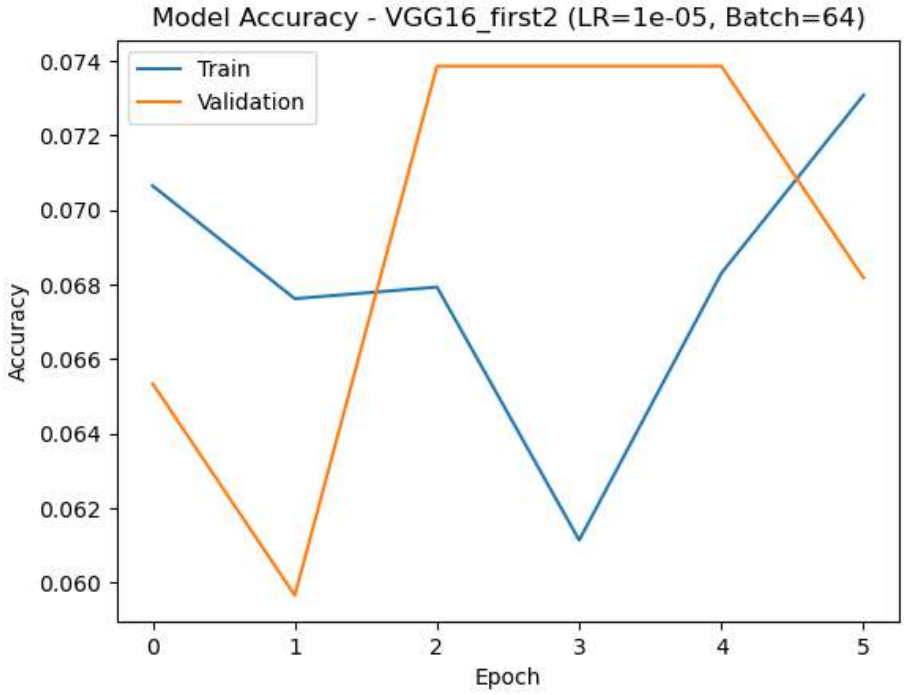
LR=0.00001 BATCH SIZE 32

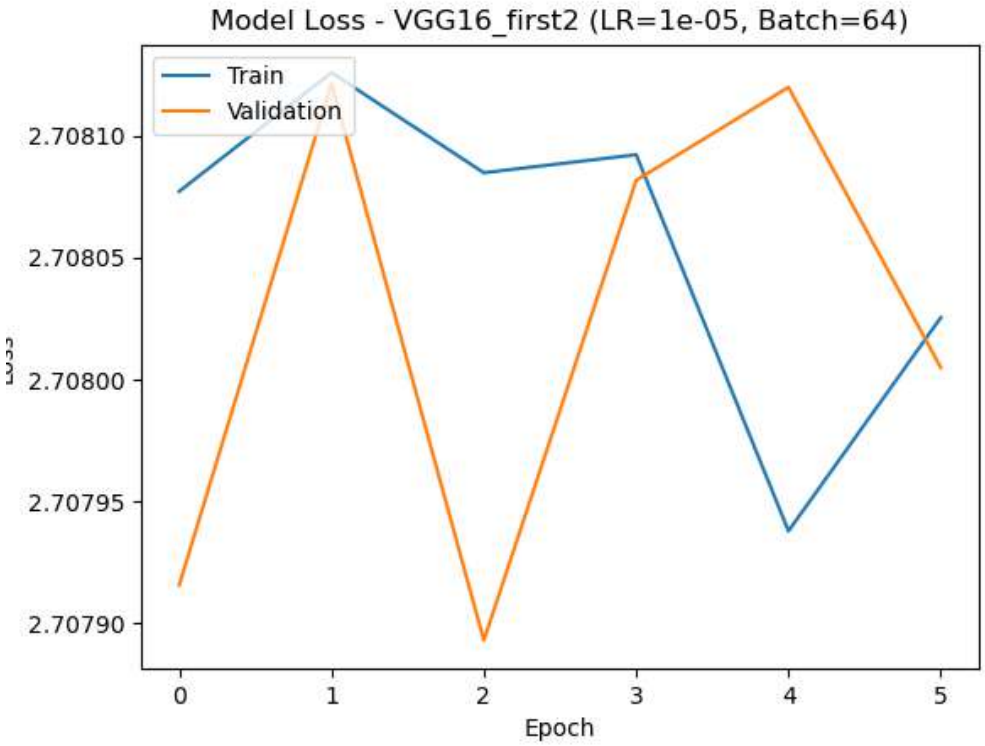




LAST 2-UNFREEZE VGG16

LR=0.00001 BATCH SIZE 64





COMMENTING ABOUT

LAST-2-UNFREEZE VGG 16

In the detailed analysis of the VGG16 model training with unfrozen first two blocks, it was discerned that the accuracies across various learning rates and batch sizes were closely aligned, with neither the training nor the validation accuracy demonstrating significant overfitting. However, the accuracies achieved were notably low, which could be indicative of underfitting or suggest potential limitations in the model's learning capacity from the data provided. The fluctuations observed in the validation accuracy might reflect a need for more consistent data or the optimization of hyperparameters to enhance the model's predictive stability and accuracy.

PART-3

!! ***Two models were constructed using the VGG16 architecture. The first model is tailored for classification tasks, where the pre-trained VGG16 base is frozen to leverage its image features, with added layers for the specific classification. The second model combines classification and regression, extending the VGG16 with a dual-head for simultaneous classification and bounding box regression. Due to the unavailability of ResNet-18 in TensorFlow and the complexity of ResNet-50 leading to overfitting, VGG16 was employed as a simpler alternative to meet the assignment's requirements.***

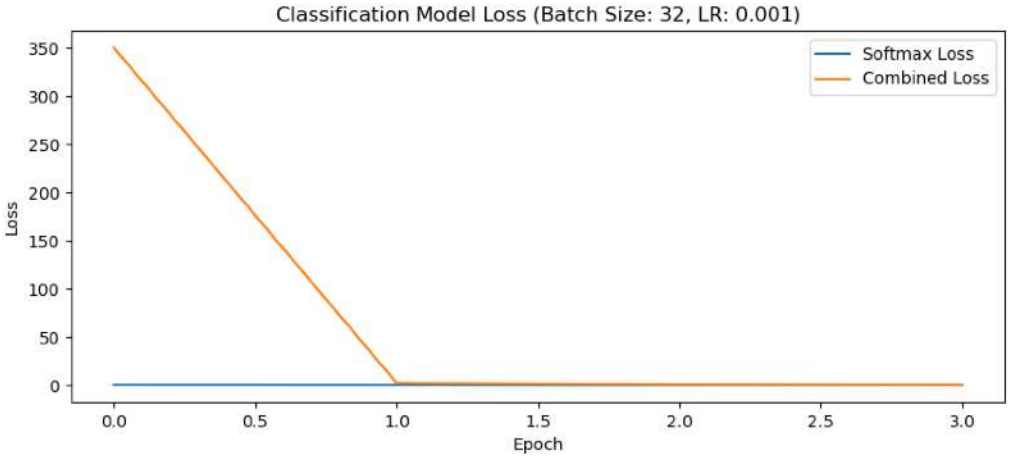
```python
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers, models

def build_vgg16_classification_and_regression_model(input_shape):
    # Load the VGG16 model, pretrained on ImageNet data
    vgg16_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)

    # Freeze the layers of the VGG16 model to prevent them from being trained
    for layer in vgg16_model.layers:
        layer.trainable = False

    # Add new layers on top of VGG16 base
    x = layers.Flatten()(vgg16_model.output)
    x = layers.Dense(4096, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(4096, activation='relu')(x)
    x = layers.Dropout(0.5)(x)

    # Classification head
    classification_output = layers.Dense(1, activation='softmax', name='classification_head')(x)

    # Regression head (for bounding box prediction)
    # This head outputs 4 values: [x, y, width, height]
    regression_output = layers.Dense(4, activation='linear', name='regression_head')(x)

    # Construct the full model with both heads
    model = models.Model(inputs=vgg16_model.input, outputs=[classification_output, regression_output])

    return model
```

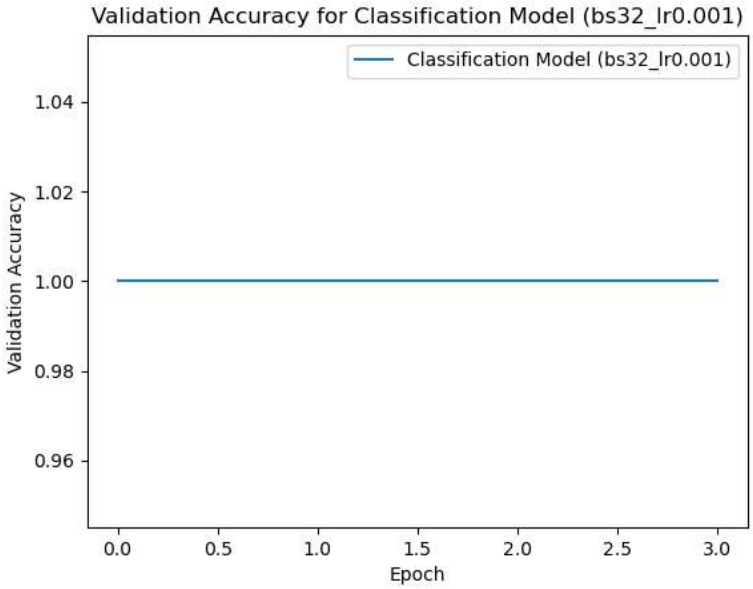
```python
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers, models

def build_vgg16_classification_model(input_shape):
    # Load the VGG16 model, pretrained on ImageNet data
    vgg16_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)

    # Freeze the layers of the VGG16 model to prevent them from being trained
    for layer in vgg16_model.layers:
        layer.trainable = False

    # Create a new model and add the VGG16 layers
    model = models.Sequential()
    model.add(vgg16_model)

    # Add new layers for your specific task
    model.add(layers.Flatten())
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(1, activation='softmax'))  # Use 'sigmoid' for binary classification

    return model
```

Python

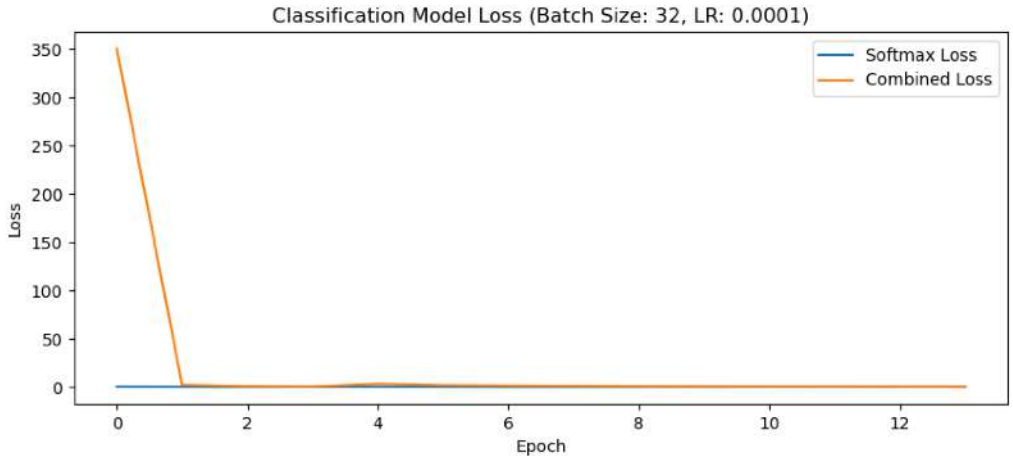
CLASSIFICATION MODEL VS BOTH HEAD MODEL

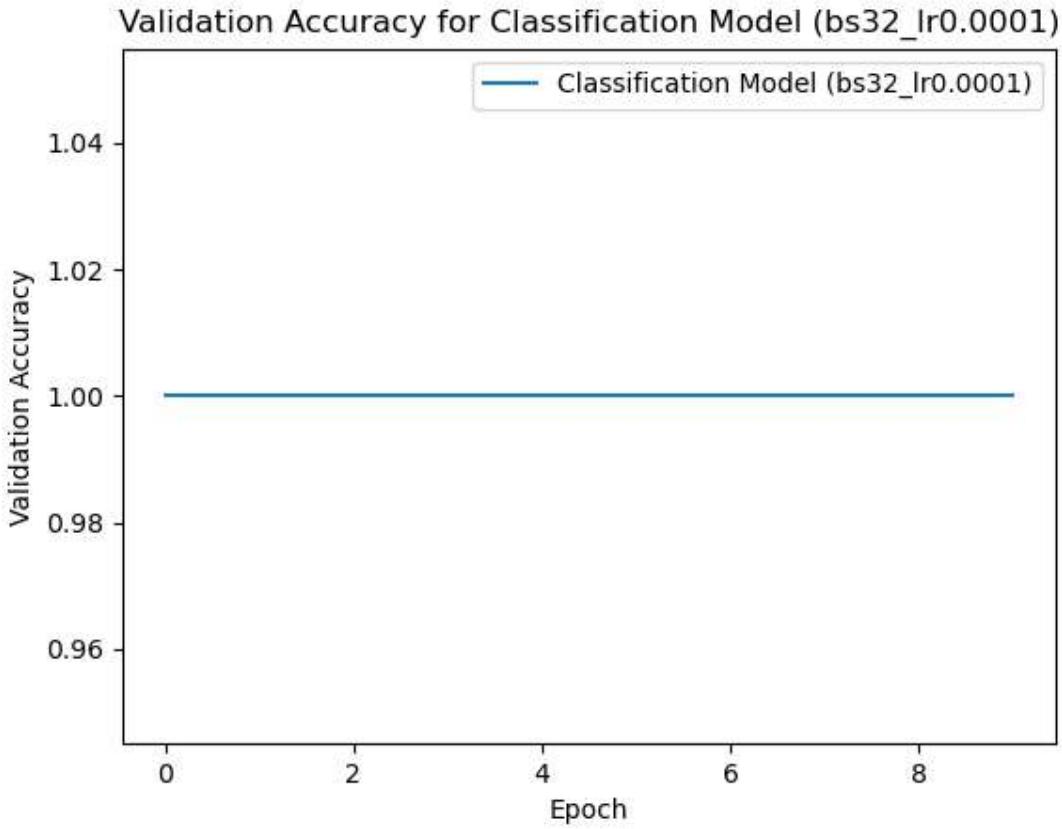
LR=0.001 BATCH SIZE 32



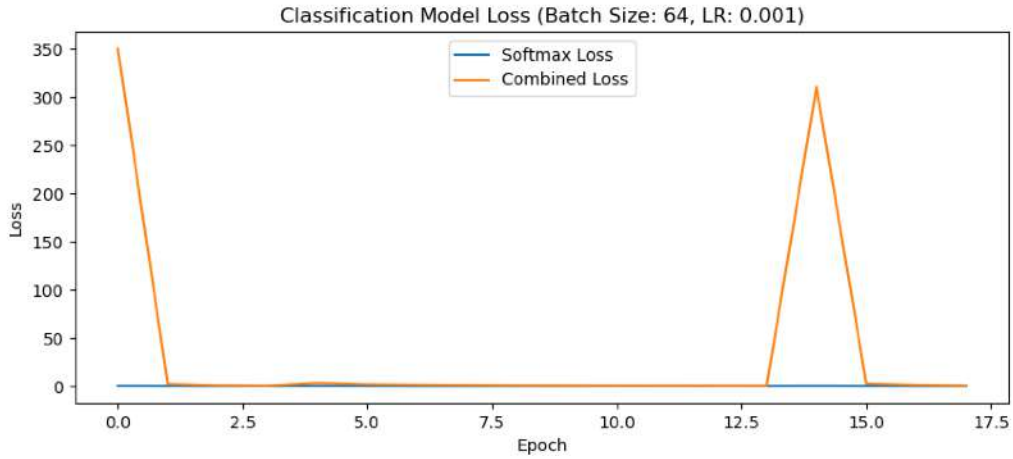


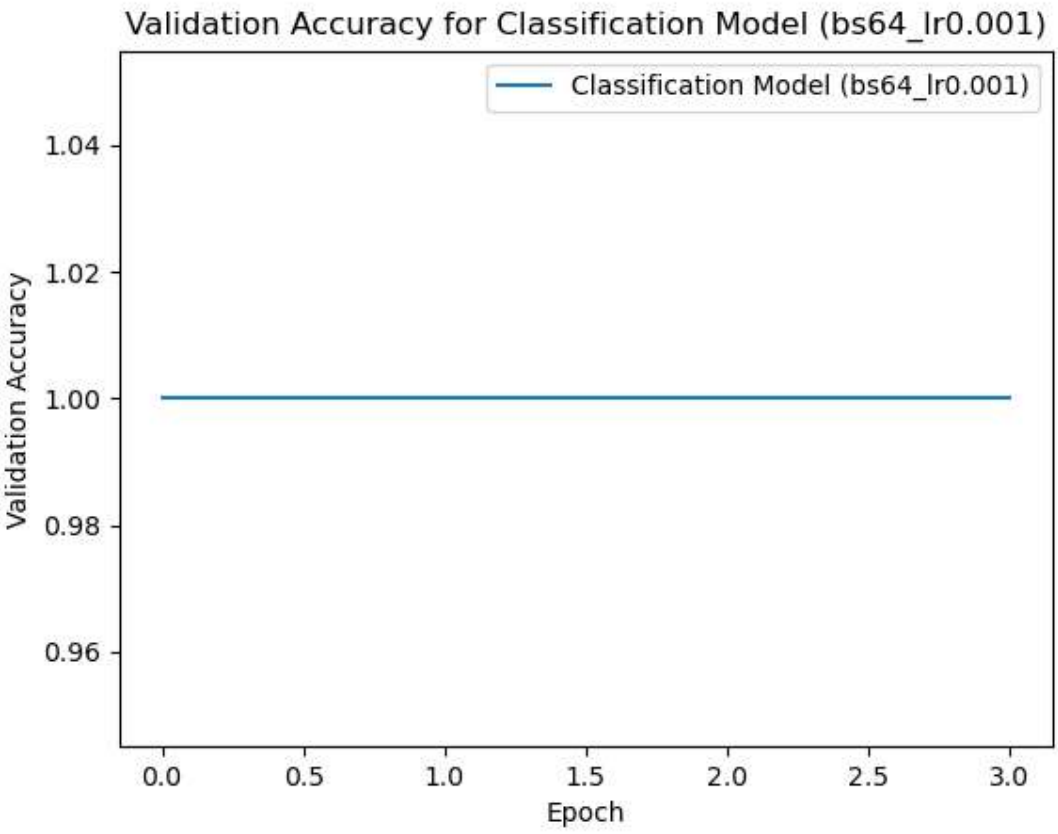
LR=0.0001 BATCH SIZE 32



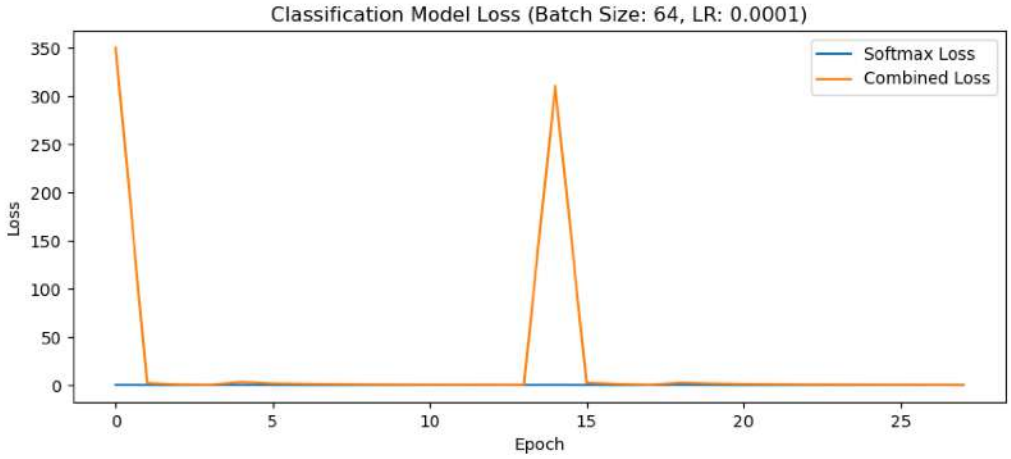


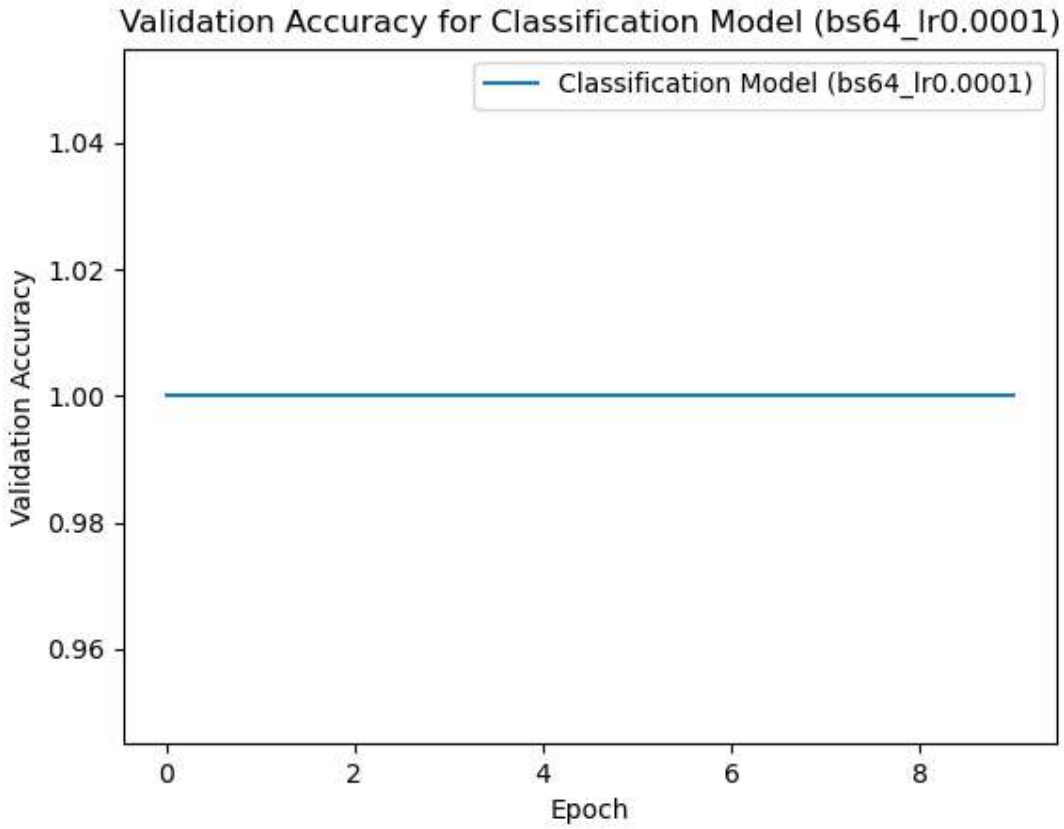
LR=0.001 BATCH SIZE 64



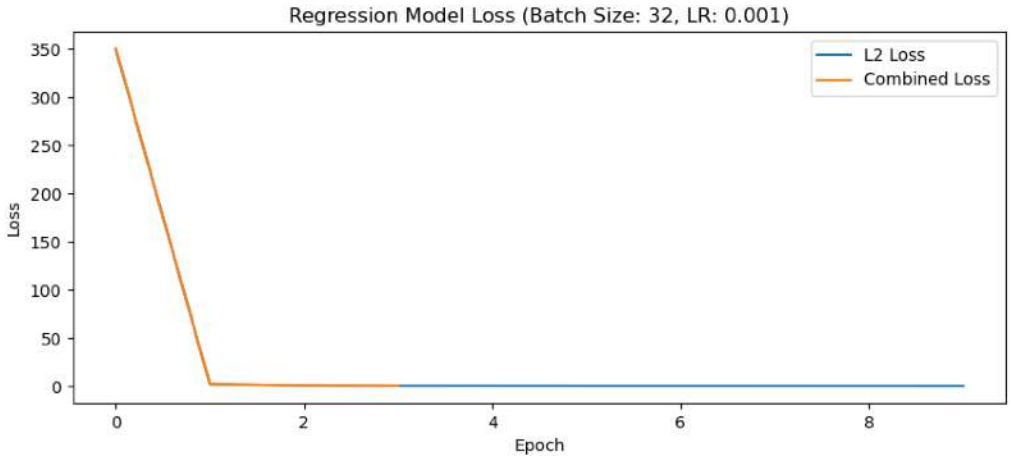


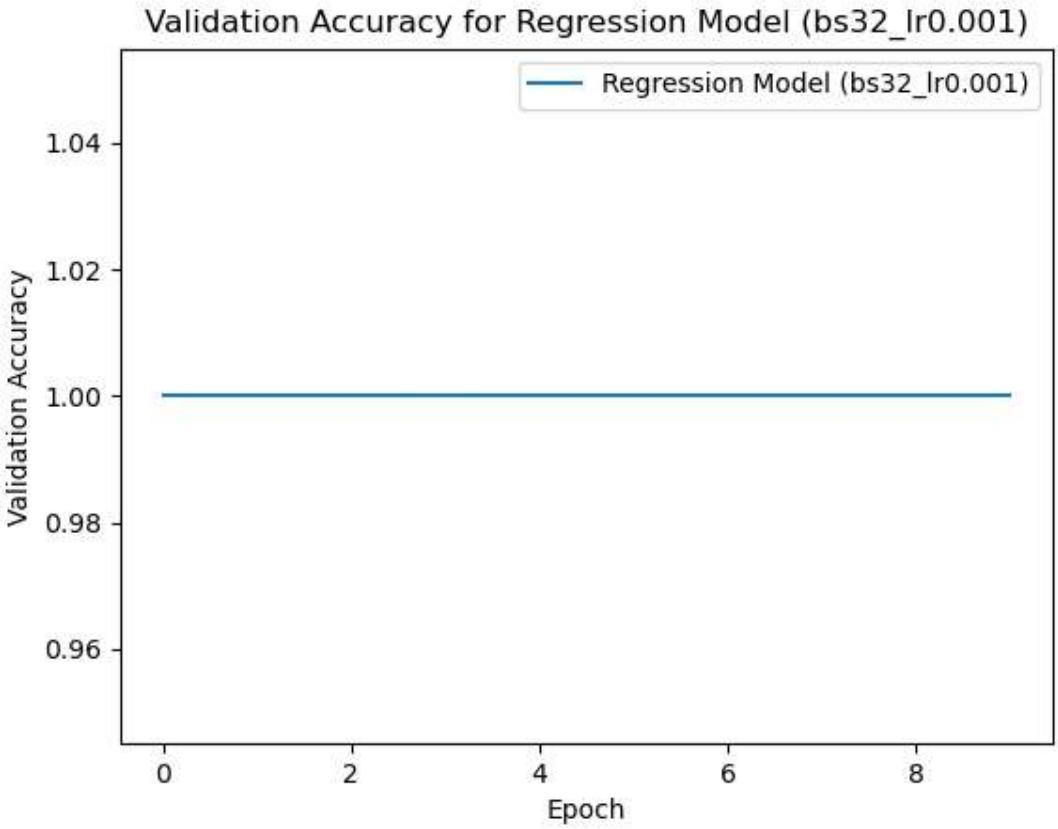
LR=0.0001 BATCH SIZE 64



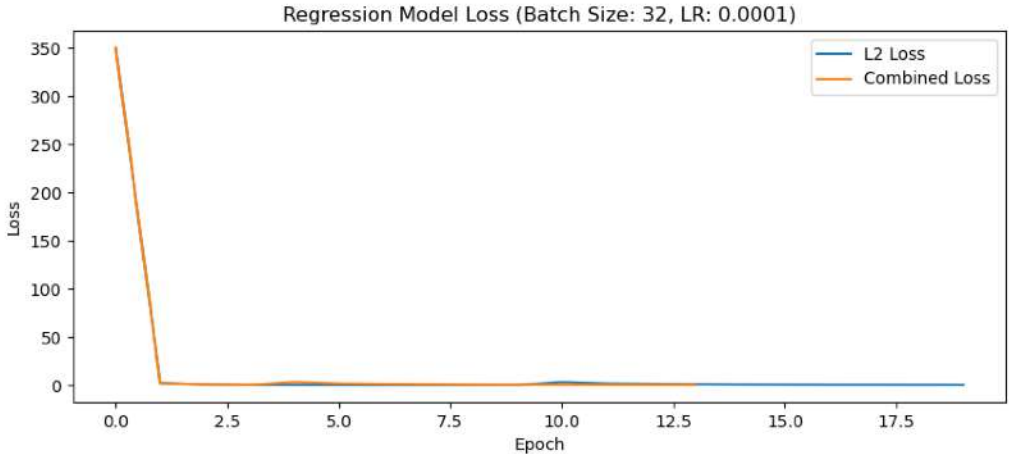


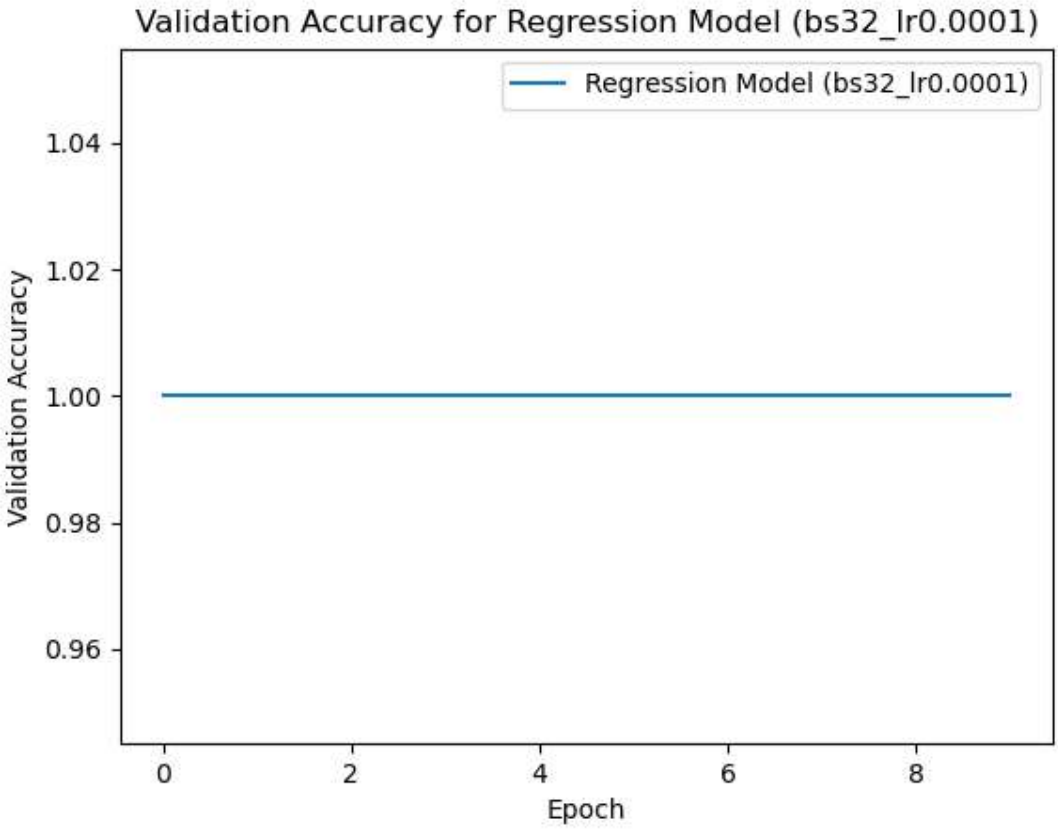
LR=0.001 BATCH SIZE 32



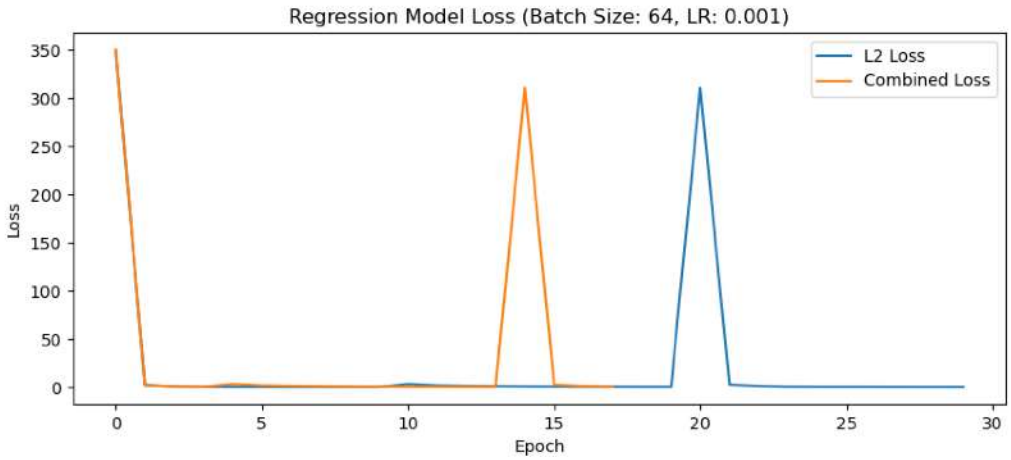


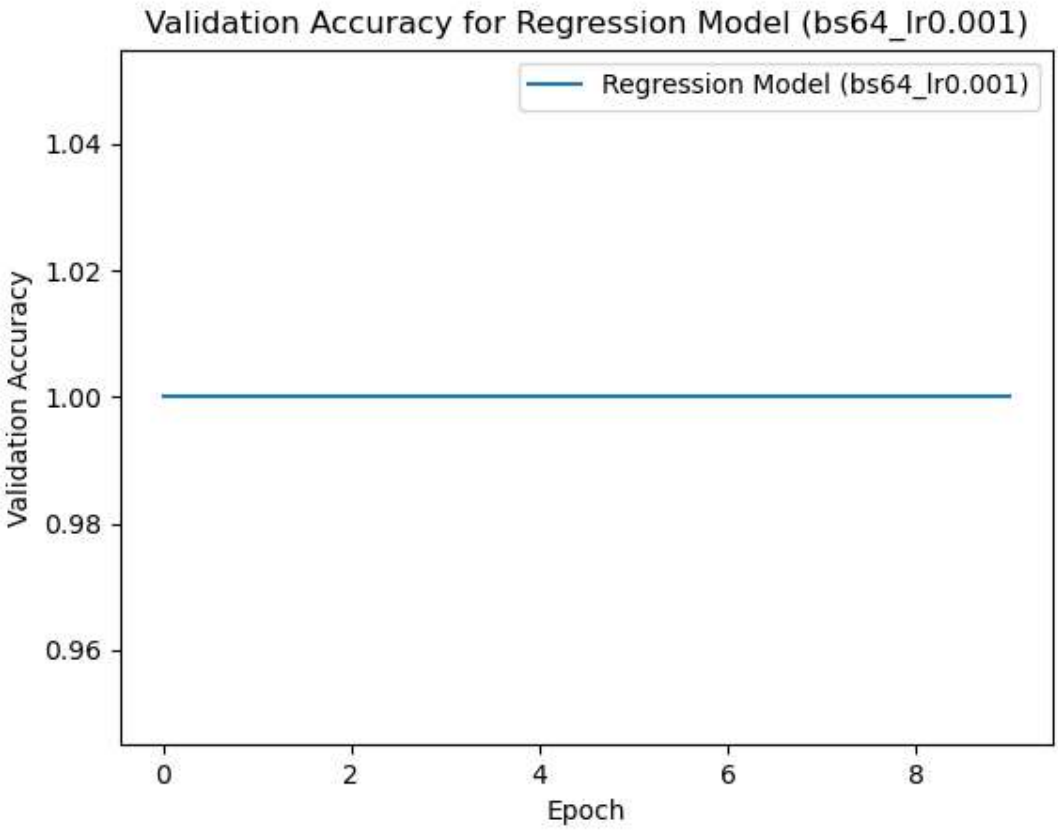
LR=0.0001 BATCH SIZE 32



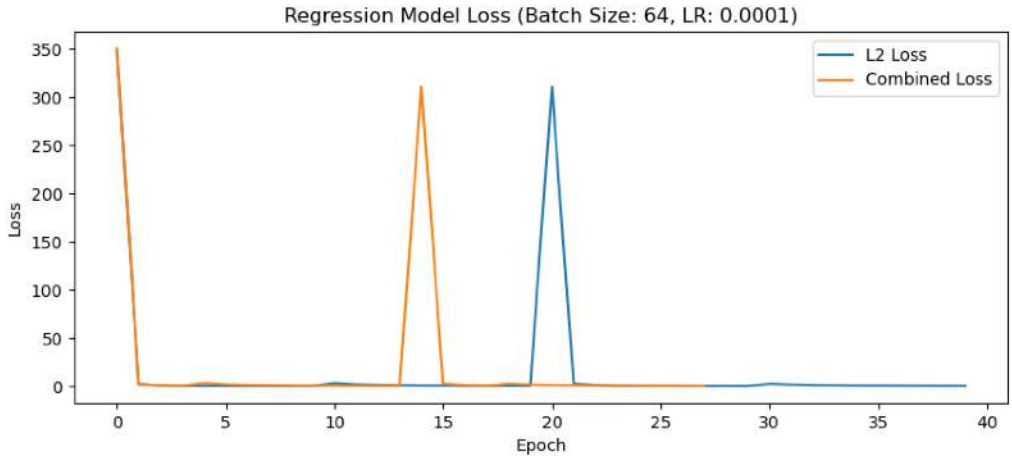


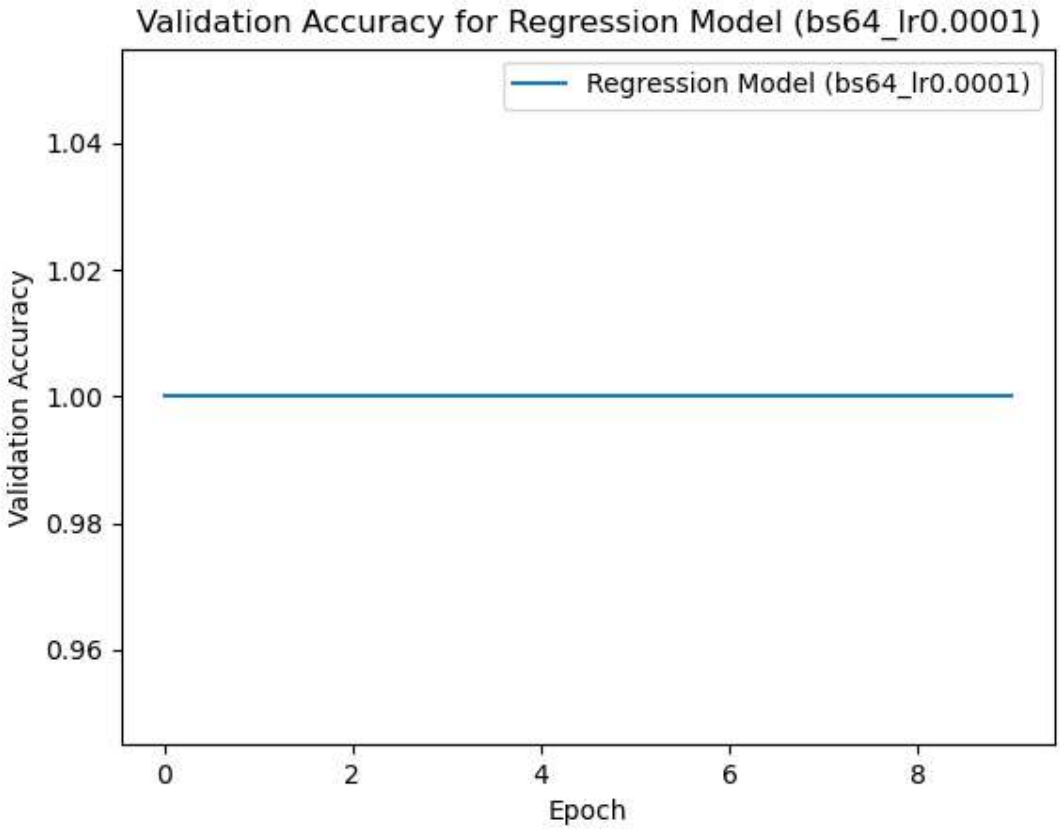
LR=0.001 BATCH SIZE 64





LR=0.0001 BATCH SIZE 64





COMMENT

The performance analysis of models with only a classification head and those with both classification and regression heads revealed varied outcomes across different learning rates and batch sizes. The classification model loss graphs indicate that changes in learning rate and batch size affected the softmax and combined losses differently, with higher learning rates generally resulting in a more pronounced decrease in loss. When both classification and regression tasks were introduced, the combined loss was impacted, showing spikes that suggest the model might be struggling to optimize both tasks simultaneously. Such spikes could indicate difficulties in learning or in the way loss is being calculated and combined for the two different tasks. The results might imply that a careful balance is required when simultaneously training for classification and regression to ensure stable convergence.