lung_cancer_kaggle-Copy1

July 11, 2024

1 Lung Cancer Image Classification

1.1 About the Dataset

1.1.1 Lung Cancer Image Dataset: A Comprehensive Collection

Explore the intricacies of lung cancer with our curated dataset, consisting of high-resolution CT scan images. This dataset is designed to aid researchers, clinicians, and machine learning/Deep learning enthusiasts in studying the diverse manifestations of lung cancer.

1.1.2 Key Features

CT Scan Images: Our dataset comprises CT scan images, providing detailed insights into lung cancer morphology. Each image is a visual representation of the complex nature of lung tumors.

Split for Comprehensive Analysis:

- **Training Set (613 Images)**: A robust training set containing 613 images meticulously labeled into four distinct classes, allowing for in-depth model training and understanding.
- **Testing Set (315 Images)**: Evaluate the model's performance on a diverse range of 315 images, each belonging to one of the four well-defined lung cancer classes.
- **Validation Set (72 Images)**: A curated validation set of 72 images, essential for fine-tuning models and ensuring generalizability.

1.1.3 Classes:

- **Class 1**: Adenocarcinoma
- **Class 2**: Large Cell Carcinoma
- **Class 3**: Normal
- **Class 4**: Squamous Cell Carcinoma

Source: https://www.kaggle.com/datasets/kabil007/lungcancer4types-imagedataset

[145]:
```
# Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from plotly.subplots import make_subplots
import plotly.graph_objects as go
```

```
import seaborn as sns
import plotly.express as px
import cv2
import warnings

import tensorflow as tf
from tensorflow.keras.regularizers import l2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, BatchNormalization,Dense,␣
↪MaxPool2D, MaxPooling2D, Flatten,GlobalMaxPooling2D, Input, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.models import Model, Sequential, Model, load_model
from tensorflow.keras.applications import ResNet50, ResNet101, ResNet152,␣
↪VGG16, VGG19, EfficientNetB0
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,␣
↪classification_report

warnings.filterwarnings("ignore")
```

[17]:

```
# Set parameters
input_size = (224,224) #  Note: pre-trained models were trained on images of␣
↪this size
batch_size=32

# Import test, train, and validation data
test_data = ImageDataGenerator().flow_from_directory(
    "./archive/Data/test",
    shuffle=False,
    batch_size = batch_size,
    target_size = input_size,
    class_mode = "categorical"
)

class_names = list(test_data.class_indices.keys())

train_data = ImageDataGenerator().flow_from_directory(
    './archive/Data/train',
    shuffle=True,
    batch_size=batch_size,
    target_size = input_size,
    class_mode = "categorical"
)

valid_data = ImageDataGenerator().flow_from_directory(
    "./archive/Data/valid",
```

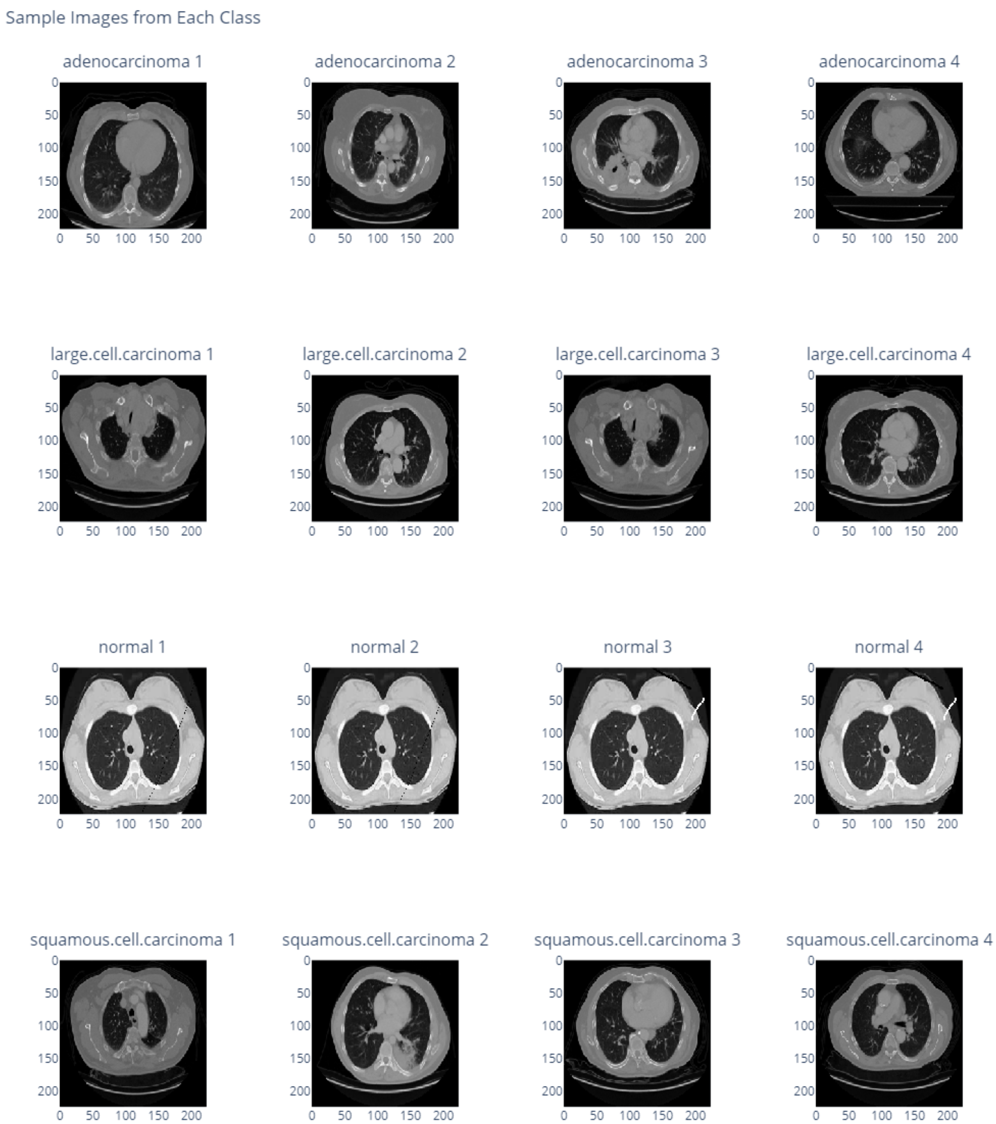
```
    shuffle=False,
    batch_size = batch_size,
    target_size = input_size,
    class_mode = "categorical"
)
```

```
Found 315 images belonging to 4 classes.
Found 613 images belonging to 4 classes.
Found 72 images belonging to 4 classes.
```

1.2 Let's take a look at images from each of the classes

[18]:
```
# Get class labels
class_labels = list(test_data.class_indices.keys())

# Function to load four images from each class
def load_images_per_class(data_gen, class_labels, num_images=4):
    images = {label: [] for label in class_labels}
    while any(len(images[label]) < num_images for label in class_labels):
        img_batch, label_batch = next(data_gen)
        for img, label in zip(img_batch, label_batch):
            class_idx = np.argmax(label)
            class_label = class_labels[class_idx]
            if len(images[class_label]) < num_images:
                images[class_label].append(img)
    return images

# Load four images per class
images_dict = load_images_per_class(test_data, class_labels, num_images=4)

# Create subplots
num_classes = len(class_labels)
fig = make_subplots(rows=num_classes, cols=4, subplot_titles=[f"{label} {i+1}"␣
 ↪for label in class_labels for i in range(4)])

# Add images to subplots
for class_idx, class_label in enumerate(class_labels):
    for img_idx, img in enumerate(images_dict[class_label]):
        fig.add_trace(
            go.Image(z=img.astype(np.uint8)),
            row=class_idx+1, col=img_idx+1
        )

# Update layout
fig.update_layout(height=300*num_classes, width=1200, title_text="Sample Images␣
 ↪from Each Class")
```

```python
# Show the plot
fig.show()
```



[51]:

```python
# Plot the training and validation accuracy for each epoch and show where the
↪highest accuracy & lowest loss are
def plot_accuracy(history):

    # Access the history data
    history_dict = history.history
```

```
    # Extract metrics
    accuracy = history_dict['accuracy']
    val_accuracy = history_dict['val_accuracy']
    loss = history_dict['loss']
    val_loss = history_dict['val_loss']
    epochs = range(1, len(accuracy) + 1)

    # Find the best validation accuracy and corresponding epoch
    best_val_acc = max(val_accuracy)
    best_val_acc_epoch = val_accuracy.index(best_val_acc) + 1

    # Find the lowest validation loss and corresponding epoch
    lowest_val_loss = min(val_loss)
    lowest_val_loss_epoch = val_loss.index(lowest_val_loss) + 1

    # Plotting the training and validation accuracy
    plt.figure(figsize=(12, 6))

    # Plot Accuracy
    plt.subplot(1, 2, 1)
    plt.plot(epochs, accuracy, label='Training Accuracy')
    plt.plot(epochs, val_accuracy, label='Validation Accuracy')
    plt.scatter(best_val_acc_epoch, best_val_acc, color='red', label=f'Best Val␣
↪Accuracy (Epoch {best_val_acc_epoch})')
    plt.text(best_val_acc_epoch, best_val_acc, f'{best_val_acc:.2f}',␣
↪color='red', ha='right')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.legend()

    # Plot Loss
    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, label='Training Loss')
    plt.plot(epochs, val_loss, label='Validation Loss')
    plt.scatter(lowest_val_loss_epoch, lowest_val_loss, color='red',␣
↪label=f'Lowest Val Loss (Epoch {lowest_val_loss_epoch})')
    plt.text(lowest_val_loss_epoch, lowest_val_loss, f'{lowest_val_loss:.2f}',␣
↪color='red', ha='right')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Training and Validation Loss')
    plt.legend()

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

[80]:
```
class_names = list(test_data.class_indices.keys())

def plot_confusion_matrix_and_report(model, test_data, class_names,␣
 ↪checkpoint_path):

    # Load the model weights from the checkpoint file
    model.load_weights(checkpoint_path)

    # Evaluate the model on the test data
    test_loss, test_accuracy = model.evaluate(test_data)

    # Generate predictions
    y_pred = model.predict(test_data)
    y_pred_classes = y_pred.argmax(axis=-1)
    y_true = test_data.classes

    # Print classification report
    report = classification_report(y_true, y_pred_classes,␣
 ↪target_names=class_names)
    print("Classification Report:\n", report)

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred_classes)

    # Plot confusion matrix using seaborn
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Purples',␣
 ↪xticklabels=class_names, yticklabels=class_names)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')

    # Rotate the x-axis labels to 45 degrees
    plt.xticks(rotation=45)

    plt.show()

    return test_accuracy, test_loss
```

2 ResNet50, 101, 152

Let's try the different ResNet models. These models increase in complexity. We will use the evaluations of these models to choose additional pre-trained models to check our image sets with. I.e., if ResNet50 outperforms ResNet152, ResNet152 may be too complex resulting in overfitting.

```
[96]: # Let's use the same learning rate for these models
learning_rate = 0.0001  # Note: we will attempt to finetune the learning rate␣
 ↪later in the notebook

# Set early stopping and checkpoints
monitor="val_loss"

early_stop = EarlyStopping(
    monitor=monitor,
    patience=10,
    restore_best_weights=True  # Restore model weights from the epoch with the␣
 ↪best value of the monitored metric
)

# The checkpoints help in case our computer crashes or our compiling is␣
 ↪interupted, but we also want to use the model from the epoch that performed␣
 ↪the best.
checkpoint_resnet50 = ModelCheckpoint(
    'resnet50_best.weights.h5',
    monitor=monitor,
    save_best_only=True,
    save_weights_only=True,
)

checkpoint_resnet101 = ModelCheckpoint(
    'resnet101_best.weights.h5',
    monitor=monitor,
    save_best_only=True,
    save_weights_only=True,
)

checkpoint_resnet152 = ModelCheckpoint(
    'resnet152_best.weights.h5',
    monitor=monitor,
    save_best_only=True,
    save_weights_only=True,
)
```

2.1 ResNet50

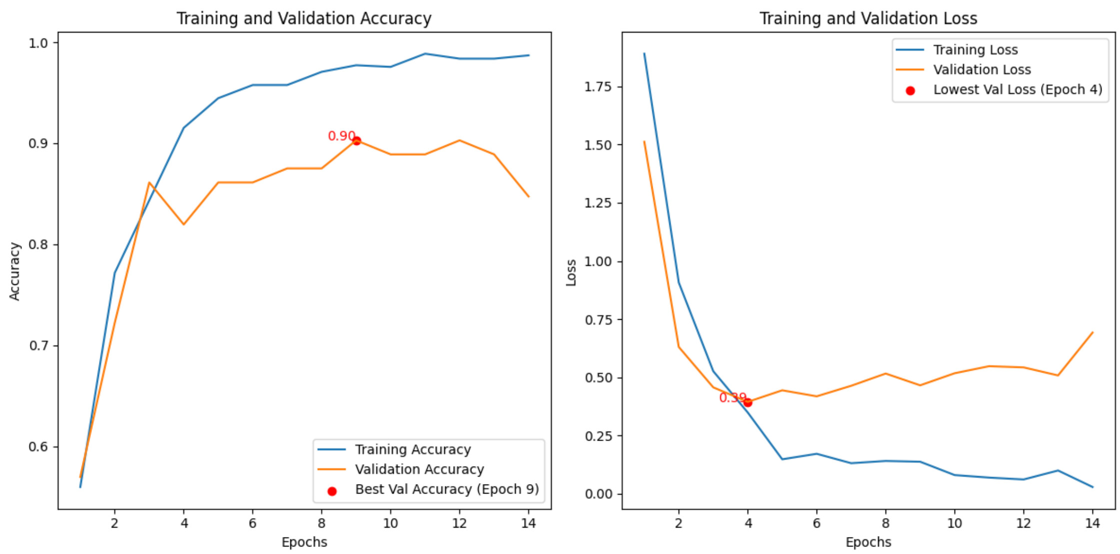
```
[43]: resnet50_model = ResNet50( include_top=False, input_shape=(224, 224, 3))
resnet50_model.trainable = False
resnet50_model = Sequential ([
    resnet50_model,
    BatchNormalization(),
    Flatten(),
    Dense(512, activation='relu'),
```

```python
    Dropout(0.3),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(4, activation='softmax')
])
resnet50_model.compile(optimizer=Adam(learning_rate),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])

history_resnet50 = resnet50_model.fit(
    train_data,
    validation_data=valid_data,
    epochs = 100,
    callbacks=[early_stop, checkpoint_resnet50],
    batch_size=batch_size,
    verbose=2
)
```

```
Epoch 1/100
20/20 - 16s - 782ms/step - accuracy: 0.5595 - loss: 1.8903 - val_accuracy:
0.5694 - val_loss: 1.5119
Epoch 2/100
20/20 - 11s - 533ms/step - accuracy: 0.7716 - loss: 0.9069 - val_accuracy:
0.7222 - val_loss: 0.6301
Epoch 3/100
20/20 - 12s - 577ms/step - accuracy: 0.8434 - loss: 0.5260 - val_accuracy:
0.8611 - val_loss: 0.4566
Epoch 4/100
20/20 - 12s - 578ms/step - accuracy: 0.9152 - loss: 0.3486 - val_accuracy:
0.8194 - val_loss: 0.3943
Epoch 5/100
20/20 - 11s - 541ms/step - accuracy: 0.9445 - loss: 0.1476 - val_accuracy:
0.8611 - val_loss: 0.4440
Epoch 6/100
20/20 - 11s - 528ms/step - accuracy: 0.9576 - loss: 0.1711 - val_accuracy:
0.8611 - val_loss: 0.4181
Epoch 7/100
20/20 - 10s - 525ms/step - accuracy: 0.9576 - loss: 0.1307 - val_accuracy:
0.8750 - val_loss: 0.4634
Epoch 8/100
20/20 - 11s - 531ms/step - accuracy: 0.9706 - loss: 0.1404 - val_accuracy:
0.8750 - val_loss: 0.5160
Epoch 9/100
20/20 - 10s - 524ms/step - accuracy: 0.9772 - loss: 0.1371 - val_accuracy:
0.9028 - val_loss: 0.4654
Epoch 10/100
20/20 - 10s - 521ms/step - accuracy: 0.9755 - loss: 0.0797 - val_accuracy:
0.8889 - val_loss: 0.5173
Epoch 11/100
```

```
20/20 - 10s - 523ms/step - accuracy: 0.9886 - loss: 0.0689 - val_accuracy:
0.8889 - val_loss: 0.5476
Epoch 12/100
20/20 - 11s - 532ms/step - accuracy: 0.9837 - loss: 0.0609 - val_accuracy:
0.9028 - val_loss: 0.5424
Epoch 13/100
20/20 - 11s - 532ms/step - accuracy: 0.9837 - loss: 0.0993 - val_accuracy:
0.8889 - val_loss: 0.5078
Epoch 14/100
20/20 - 11s - 535ms/step - accuracy: 0.9869 - loss: 0.0285 - val_accuracy:
0.8472 - val_loss: 0.6921
```

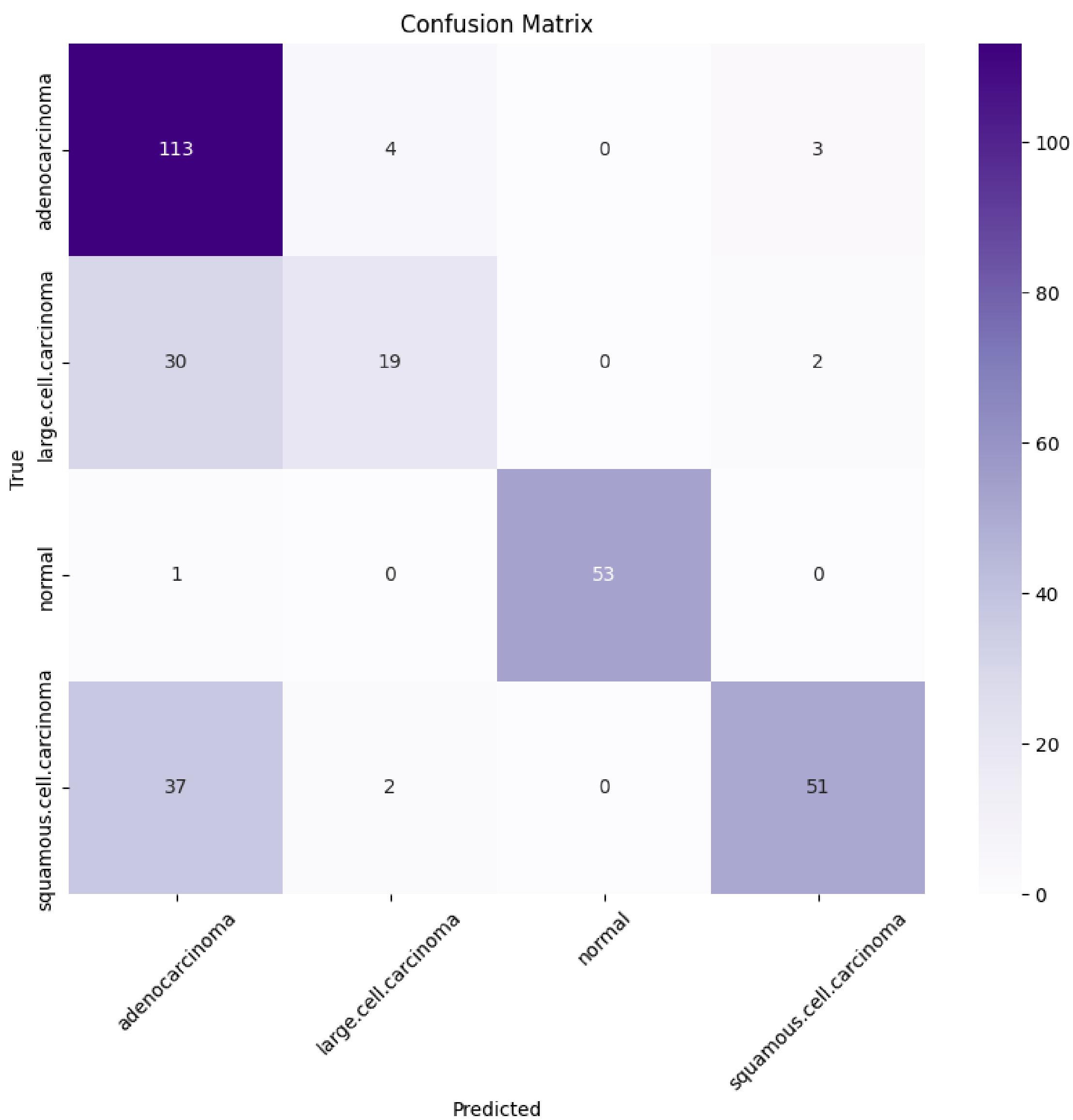
[60]:
```
plot_accuracy(history_resnet50)
```



[81]:
```
resnet50_acc, resnet50_loss = plot_confusion_matrix_and_report(resnet50_model,
    test_data, class_names, './resnet50_best.weights.h5')
```

```
10/10            3s 315ms/step -
accuracy: 0.8408 - loss: 0.4892
10/10            3s 311ms/step
Classification Report:
                         precision    recall  f1-score   support

         adenocarcinoma       0.62      0.94      0.75       120
   large.cell.carcinoma       0.76      0.37      0.50        51
                 normal       1.00      0.98      0.99        54
squamous.cell.carcinoma       0.91      0.57      0.70        90

               accuracy                           0.75       315
```

```
   macro avg       0.82      0.72      0.74       315
weighted avg       0.79      0.75      0.74       315
```



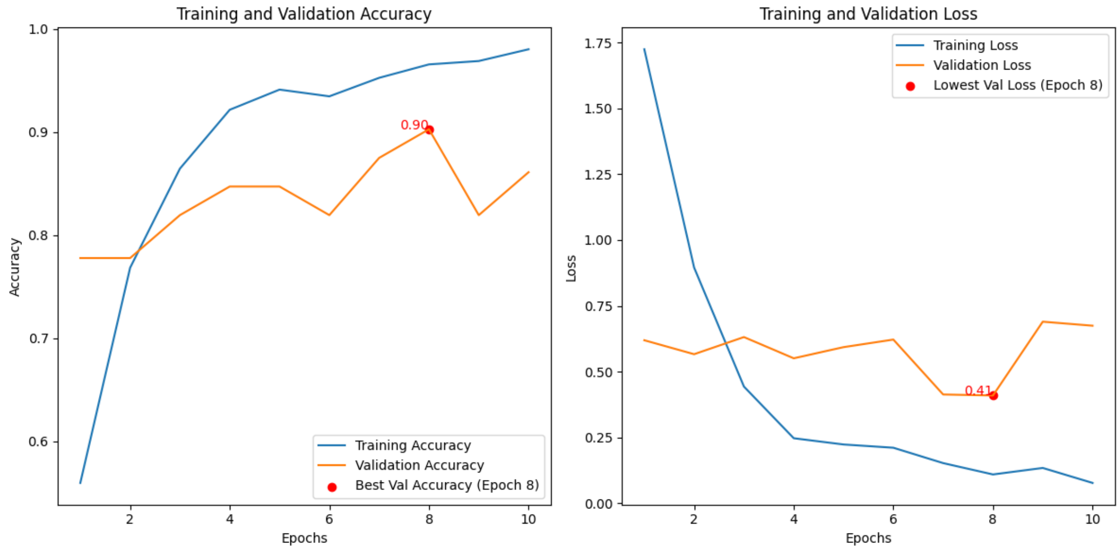
2.2 ResNet101

[79]:
```
resnet101_model = ResNet101( include_top=False, input_shape=(224, 224, 3))
resnet101_model.trainable = False
resnet101_model = Sequential ([
    resnet101_model,
    BatchNormalization(),
    Flatten(),
```

```
    Dense(512, activation='relu'),
    Dropout(0.3),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(4, activation='softmax')
])
resnet101_model.compile(optimizer=Adam(learning_rate),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])

history_resnet101 = resnet101_model.fit(
    train_data,
    validation_data=valid_data,
    epochs = 100,
    callbacks=[early_stop, checkpoint_resnet101],
    batch_size=batch_size,
    verbose=2
)
```

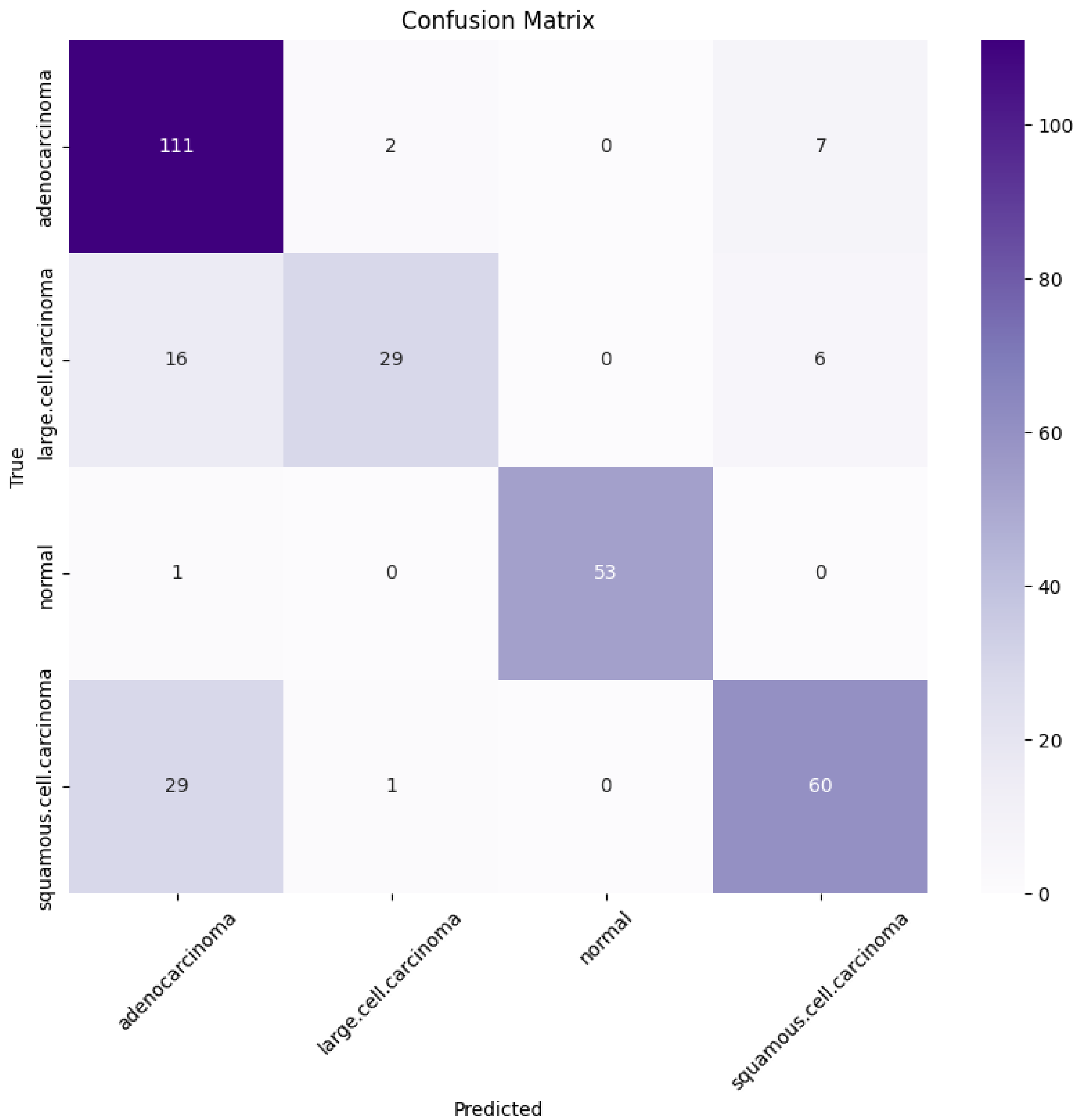
```
Epoch 1/100
20/20 - 25s - 1s/step - accuracy: 0.5595 - loss: 1.7252 - val_accuracy: 0.7778 -
val_loss: 0.6190
Epoch 2/100
20/20 - 18s - 885ms/step - accuracy: 0.7684 - loss: 0.8953 - val_accuracy:
0.7778 - val_loss: 0.5659
Epoch 3/100
20/20 - 17s - 851ms/step - accuracy: 0.8646 - loss: 0.4430 - val_accuracy:
0.8194 - val_loss: 0.6311
Epoch 4/100
20/20 - 18s - 902ms/step - accuracy: 0.9217 - loss: 0.2465 - val_accuracy:
0.8472 - val_loss: 0.5504
Epoch 5/100
20/20 - 17s - 856ms/step - accuracy: 0.9413 - loss: 0.2228 - val_accuracy:
0.8472 - val_loss: 0.5927
Epoch 6/100
20/20 - 16s - 820ms/step - accuracy: 0.9347 - loss: 0.2104 - val_accuracy:
0.8194 - val_loss: 0.6216
Epoch 7/100
20/20 - 17s - 867ms/step - accuracy: 0.9527 - loss: 0.1520 - val_accuracy:
0.8750 - val_loss: 0.4128
Epoch 8/100
20/20 - 17s - 870ms/step - accuracy: 0.9657 - loss: 0.1088 - val_accuracy:
0.9028 - val_loss: 0.4090
Epoch 9/100
20/20 - 16s - 807ms/step - accuracy: 0.9690 - loss: 0.1336 - val_accuracy:
0.8194 - val_loss: 0.6896
Epoch 10/100
20/20 - 16s - 813ms/step - accuracy: 0.9804 - loss: 0.0766 - val_accuracy:
0.8611 - val_loss: 0.6743
```

[82]:
```
plot_accuracy(history_resnet101)
```



[83]:
```
resnet101_acc, resnet101_loss =␣
↪plot_confusion_matrix_and_report(resnet101_model, test_data, class_names, './
↪resnet101_best.weights.h5')
```

```
10/10            6s 548ms/step -
accuracy: 0.8572 - loss: 0.4886
10/10            9s 720ms/step
Classification Report:
                      precision    recall  f1-score   support

       adenocarcinoma       0.71      0.93      0.80       120
  large.cell.carcinoma       0.91      0.57      0.70        51
               normal       1.00      0.98      0.99        54
squamous.cell.carcinoma       0.82      0.67      0.74        90

             accuracy                           0.80       315
            macro avg       0.86      0.79      0.81       315
         weighted avg       0.82      0.80      0.80       315
```



2.3 ResNet152

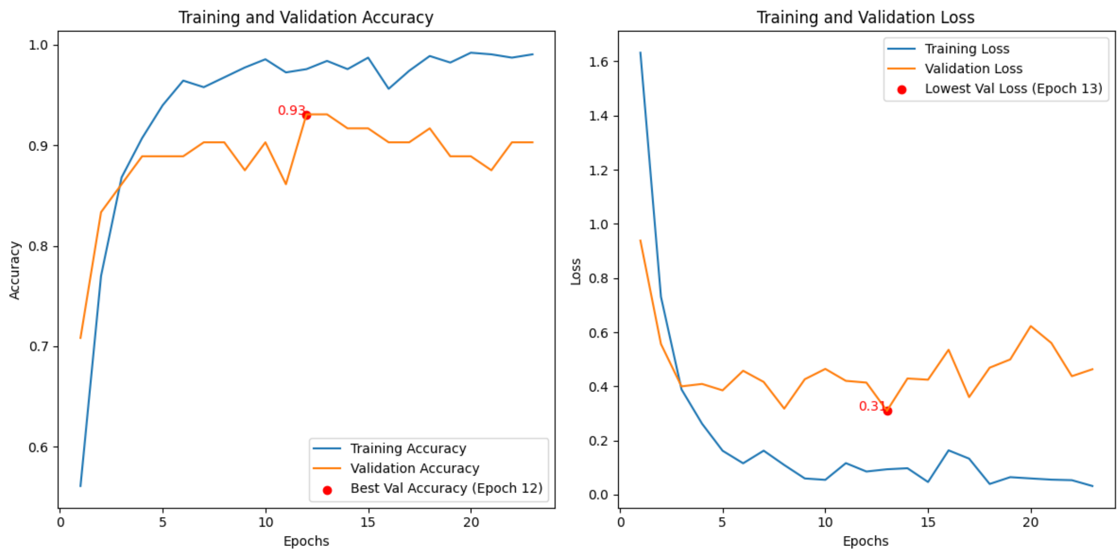
[86]:
```
resnet152_model = ResNet152( include_top=False, input_shape=(224, 224, 3))
resnet152_model.trainable = False
resnet152_model = Sequential ([
    resnet152_model,
    BatchNormalization(),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.3),
    Dense(256, activation='relu'),
```

```
    Dropout(0.5),
    Dense(4, activation='softmax')
])
resnet152_model.compile(optimizer=Adam(learning_rate),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])

history_resnet152 = resnet152_model.fit(
    train_data,
    validation_data=valid_data,
    epochs = 100,
    callbacks=[early_stop, checkpoint_resnet152],
    batch_size=batch_size,
    verbose=2
)
```

```
Epoch 1/100
20/20 - 34s - 2s/step - accuracy: 0.5612 - loss: 1.6322 - val_accuracy: 0.7083 -
val_loss: 0.9383
Epoch 2/100
20/20 - 24s - 1s/step - accuracy: 0.7700 - loss: 0.7305 - val_accuracy: 0.8333 -
val_loss: 0.5562
Epoch 3/100
20/20 - 25s - 1s/step - accuracy: 0.8679 - loss: 0.3887 - val_accuracy: 0.8611 -
val_loss: 0.3998
Epoch 4/100
20/20 - 23s - 1s/step - accuracy: 0.9070 - loss: 0.2619 - val_accuracy: 0.8889 -
val_loss: 0.4088
Epoch 5/100
20/20 - 24s - 1s/step - accuracy: 0.9396 - loss: 0.1621 - val_accuracy: 0.8889 -
val_loss: 0.3851
Epoch 6/100
20/20 - 23s - 1s/step - accuracy: 0.9641 - loss: 0.1157 - val_accuracy: 0.8889 -
val_loss: 0.4574
Epoch 7/100
20/20 - 23s - 1s/step - accuracy: 0.9576 - loss: 0.1623 - val_accuracy: 0.9028 -
val_loss: 0.4162
Epoch 8/100
20/20 - 24s - 1s/step - accuracy: 0.9674 - loss: 0.1094 - val_accuracy: 0.9028 -
val_loss: 0.3176
Epoch 9/100
20/20 - 23s - 1s/step - accuracy: 0.9772 - loss: 0.0597 - val_accuracy: 0.8750 -
val_loss: 0.4261
Epoch 10/100
20/20 - 23s - 1s/step - accuracy: 0.9853 - loss: 0.0546 - val_accuracy: 0.9028 -
val_loss: 0.4643
Epoch 11/100
20/20 - 24s - 1s/step - accuracy: 0.9723 - loss: 0.1163 - val_accuracy: 0.8611 -
val_loss: 0.4202
```

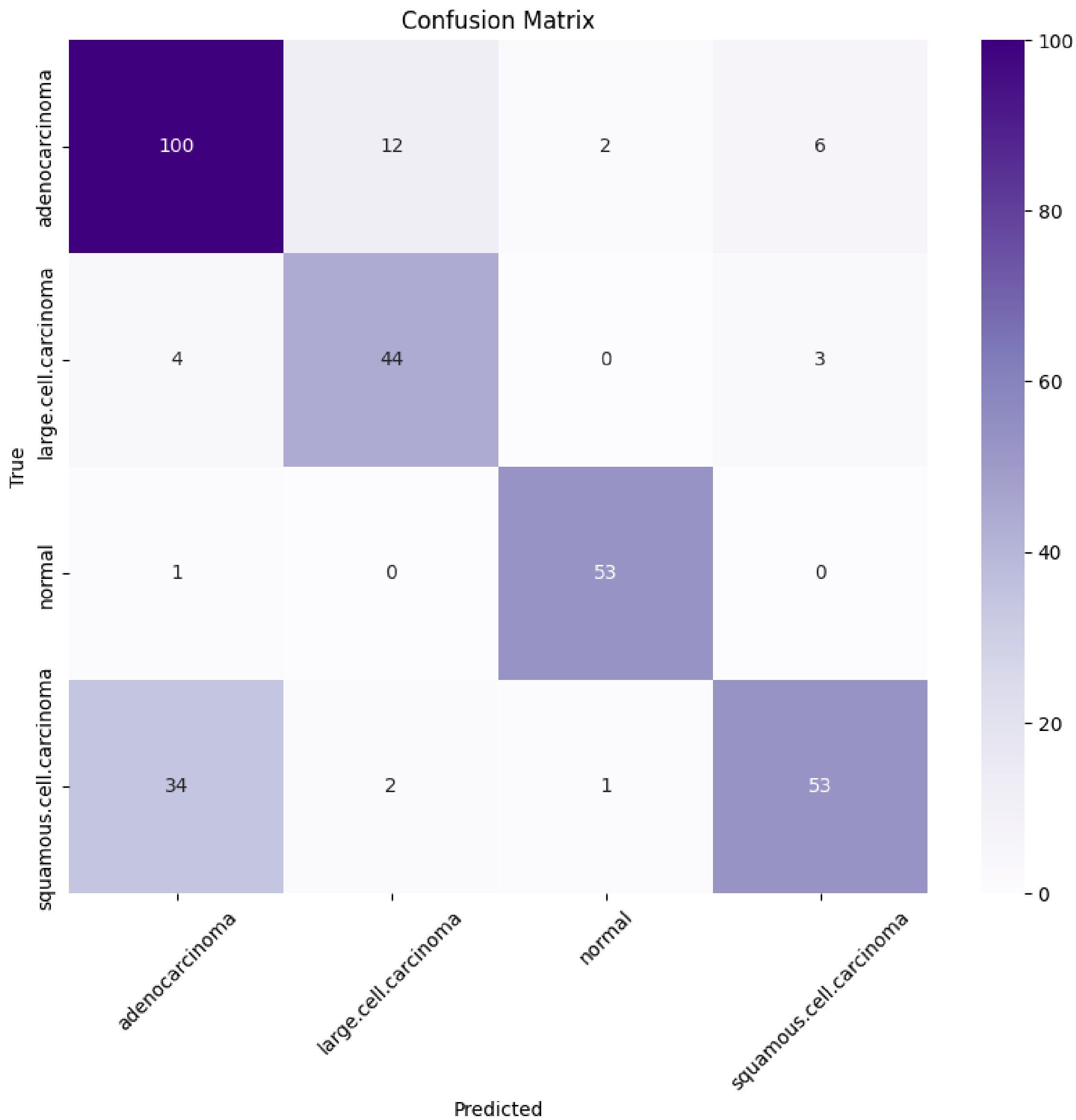
```
Epoch 12/100
20/20 - 23s - 1s/step - accuracy: 0.9755 - loss: 0.0853 - val_accuracy: 0.9306 -
val_loss: 0.4138
Epoch 13/100
20/20 - 24s - 1s/step - accuracy: 0.9837 - loss: 0.0936 - val_accuracy: 0.9306 -
val_loss: 0.3097
Epoch 14/100
20/20 - 23s - 1s/step - accuracy: 0.9755 - loss: 0.0975 - val_accuracy: 0.9167 -
val_loss: 0.4290
Epoch 15/100
20/20 - 23s - 1s/step - accuracy: 0.9869 - loss: 0.0466 - val_accuracy: 0.9167 -
val_loss: 0.4244
Epoch 16/100
20/20 - 23s - 1s/step - accuracy: 0.9560 - loss: 0.1635 - val_accuracy: 0.9028 -
val_loss: 0.5351
Epoch 17/100
20/20 - 23s - 1s/step - accuracy: 0.9739 - loss: 0.1328 - val_accuracy: 0.9028 -
val_loss: 0.3600
Epoch 18/100
20/20 - 22s - 1s/step - accuracy: 0.9886 - loss: 0.0393 - val_accuracy: 0.9167 -
val_loss: 0.4688
Epoch 19/100
20/20 - 22s - 1s/step - accuracy: 0.9821 - loss: 0.0645 - val_accuracy: 0.8889 -
val_loss: 0.4993
Epoch 20/100
20/20 - 22s - 1s/step - accuracy: 0.9918 - loss: 0.0597 - val_accuracy: 0.8889 -
val_loss: 0.6225
Epoch 21/100
20/20 - 23s - 1s/step - accuracy: 0.9902 - loss: 0.0553 - val_accuracy: 0.8750 -
val_loss: 0.5602
Epoch 22/100
20/20 - 22s - 1s/step - accuracy: 0.9869 - loss: 0.0533 - val_accuracy: 0.9028 -
val_loss: 0.4374
Epoch 23/100
20/20 - 22s - 1s/step - accuracy: 0.9902 - loss: 0.0318 - val_accuracy: 0.9028 -
val_loss: 0.4630
```

[88]:
```
plot_accuracy(history_resnet152)
```



[89]:
```
resnet152_acc, resnet152_loss =␣
↪plot_confusion_matrix_and_report(resnet152_model, test_data, class_names, './
↪resnet152_best.weights.h5')
```

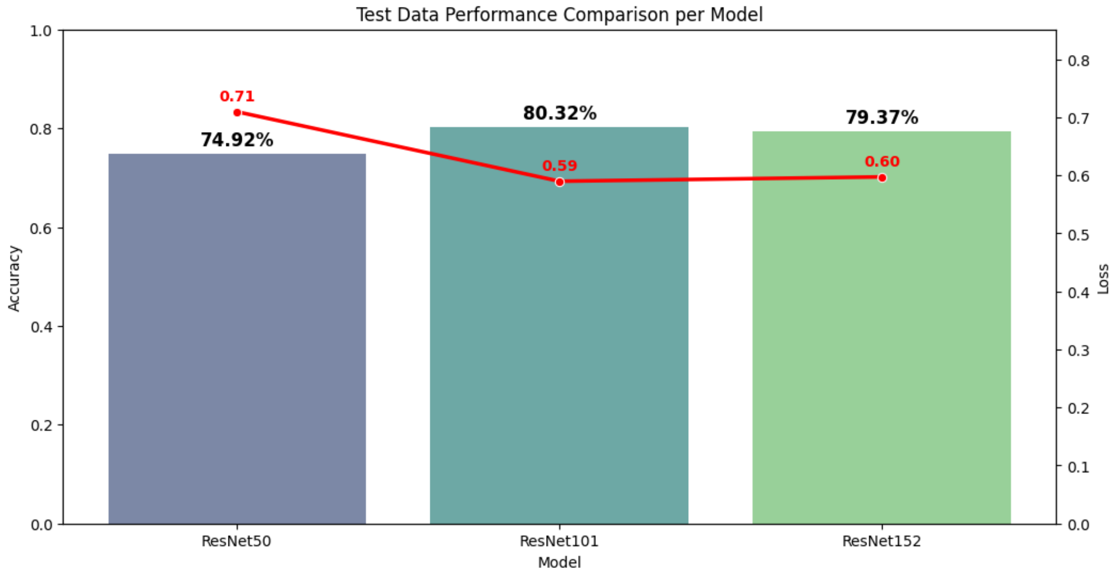
```
10/10              8s 782ms/step -
accuracy: 0.8474 - loss: 0.4727
10/10              13s 1s/step
Classification Report:
                         precision    recall  f1-score   support

         adenocarcinoma       0.72      0.83      0.77       120
    large.cell.carcinoma       0.76      0.86      0.81        51
                 normal       0.95      0.98      0.96        54
squamous.cell.carcinoma       0.85      0.59      0.70        90

               accuracy                           0.79       315
              macro avg       0.82      0.82      0.81       315
           weighted avg       0.80      0.79      0.79       315
```



2.4 Let's see which ResNet model performed the best

[118]:
```
def plot_comparison(model_names, accuracies, losses):

    # Create a DataFrame from the lists for easy plotting
    data = {
        'Model': model_names,
        'Accuracy': accuracies,
        'Loss': losses
    }
    df = pd.DataFrame(data)
```

```
    # Next, create a bar plot for accuracies and losses
    fig, ax1 = plt.subplots(figsize=(12, 6))

    # Bar plot for accuracies
    sns.barplot(x='Model', y='Accuracy', data=df, ax=ax1, palette='viridis',␣
↪alpha=0.7)
    ax1.set_ylabel('Accuracy')
    ax1.set_ylim(0, 1)
    ax1.set_title('Test Data Performance Comparison per Model')

    # Add accuracy values on top of the bars (note, higher accuracy is better)
    for p in ax1.patches:
        ax1.annotate(f'{p.get_height() * 100:.2f}%',
                     (p.get_x() + p.get_width() / 2., p.get_height()),
                     ha='center', va='center',
                     xytext=(0, 9),
                     textcoords='offset points',
                     color='black', fontsize=12, fontweight='bold')

    # Create a secondary y-axis for losses
    ax2 = ax1.twinx()
    sns.lineplot(x='Model', y='Loss', data=df, ax=ax2, color='red', marker='o',␣
↪linewidth=2.5)
    ax2.set_ylabel('Loss')
    ax2.set_ylim(0, max(losses) * 1.2)

    # Add loss values above the points on the line plot (note, lower loss is␣
↪better)
    for line in ax2.lines:
        for x, y in zip(line.get_xdata(), line.get_ydata()):
            ax2.annotate(f'{y:.2f}',
                         (x, y),
                         ha='center', va='bottom',
                         xytext=(0, 5),
                         textcoords='offset points',
                         color='red', fontsize=10, fontweight='bold')

    # Display the plot
    plt.show()

resnet_models = ["ResNet50","ResNet101","ResNet152"]
resnet_accs = [resnet50_acc, resnet101_acc, resnet152_acc]
resnet_loss = [resnet50_loss, resnet101_loss, resnet152_loss]

plot_comparison(resnet_models, resnet_accs, resnet_loss)
```



2.5 Looks like our ResNet101 Model performed the best out of the ResNet Models!

Let's try VGG16, VGG19, and EfficientNetB0. We will take the best of the six models and see if we can hypertune some of their parameters.

[97]:
```
# Let's use the same learning rate for these models
learning_rate = 0.0001

# Set early stopping and checkpoints
monitor="val_loss"

early_stop = EarlyStopping(
    monitor=monitor,
    patience=10,
    restore_best_weights=True  # Restore model weights from the epoch with the␣
 ↪best value of the monitored metric
)

# The checkpoints help in case our computer crashes or our compiling is␣
 ↪interupted, but we also want to use the model from the epoch that performed␣
 ↪the best.
checkpoint_VGG16 = ModelCheckpoint(
    'VGG16_best.weights.h5',
    monitor=monitor,
    save_best_only=True,
    save_weights_only=True,
)
```

```
checkpoint_VGG19 = ModelCheckpoint(
    'VGG19_best.weights.h5',
    monitor=monitor,
    save_best_only=True,
    save_weights_only=True,
)

checkpoint_EfficientNetB0 = ModelCheckpoint(
    'EfficientNetB0_best.weights.h5',
    monitor=monitor,
    save_best_only=True,
    save_weights_only=True,
)
```

2.6 VGG16

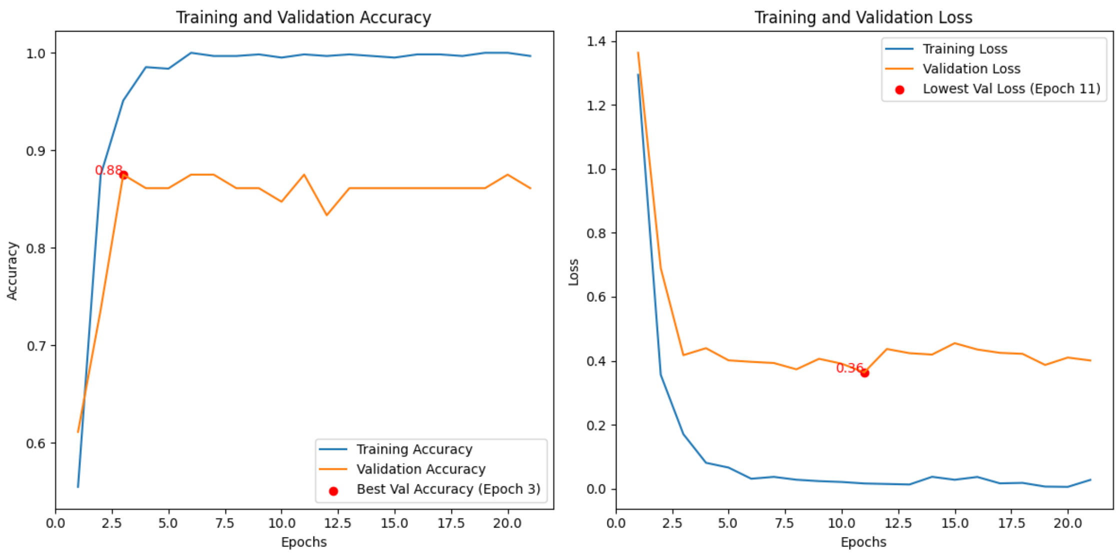
[98]:
```
VGG16_model = VGG16( include_top=False, input_shape=(224, 224, 3))
VGG16_model.trainable = False
VGG16_model = Sequential([
    VGG16_model,
    BatchNormalization(),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(4, activation='softmax')
])
VGG16_model.compile(optimizer=Adam(learning_rate),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])

history_VGG16 = VGG16_model.fit(
    train_data,
    validation_data=valid_data,
    epochs = 100,
    callbacks=[early_stop, checkpoint_VGG16],
    batch_size=batch_size,
    verbose=2
)
```

```
Epoch 1/100
20/20 - 17s - 828ms/step - accuracy: 0.5546 - loss: 1.2943 - val_accuracy:
0.6111 - val_loss: 1.3630
Epoch 2/100
20/20 - 19s - 969ms/step - accuracy: 0.8744 - loss: 0.3559 - val_accuracy:
0.7361 - val_loss: 0.6896
Epoch 3/100
20/20 - 19s - 959ms/step - accuracy: 0.9511 - loss: 0.1704 - val_accuracy:
0.8750 - val_loss: 0.4174
```

```
Epoch 4/100
20/20 - 19s - 957ms/step - accuracy: 0.9853 - loss: 0.0808 - val_accuracy:
0.8611 - val_loss: 0.4392
Epoch 5/100
20/20 - 19s - 955ms/step - accuracy: 0.9837 - loss: 0.0658 - val_accuracy:
0.8611 - val_loss: 0.4012
Epoch 6/100
20/20 - 20s - 989ms/step - accuracy: 1.0000 - loss: 0.0310 - val_accuracy:
0.8750 - val_loss: 0.3966
Epoch 7/100
20/20 - 19s - 964ms/step - accuracy: 0.9967 - loss: 0.0367 - val_accuracy:
0.8750 - val_loss: 0.3928
Epoch 8/100
20/20 - 19s - 957ms/step - accuracy: 0.9967 - loss: 0.0278 - val_accuracy:
0.8611 - val_loss: 0.3731
Epoch 9/100
20/20 - 19s - 947ms/step - accuracy: 0.9984 - loss: 0.0235 - val_accuracy:
0.8611 - val_loss: 0.4060
Epoch 10/100
20/20 - 19s - 961ms/step - accuracy: 0.9951 - loss: 0.0208 - val_accuracy:
0.8472 - val_loss: 0.3908
Epoch 11/100
20/20 - 19s - 949ms/step - accuracy: 0.9984 - loss: 0.0161 - val_accuracy:
0.8750 - val_loss: 0.3637
Epoch 12/100
20/20 - 19s - 935ms/step - accuracy: 0.9967 - loss: 0.0145 - val_accuracy:
0.8333 - val_loss: 0.4367
Epoch 13/100
20/20 - 19s - 950ms/step - accuracy: 0.9984 - loss: 0.0129 - val_accuracy:
0.8611 - val_loss: 0.4236
Epoch 14/100
20/20 - 19s - 942ms/step - accuracy: 0.9967 - loss: 0.0369 - val_accuracy:
0.8611 - val_loss: 0.4193
Epoch 15/100
20/20 - 18s - 923ms/step - accuracy: 0.9951 - loss: 0.0277 - val_accuracy:
0.8611 - val_loss: 0.4549
Epoch 16/100
20/20 - 19s - 929ms/step - accuracy: 0.9984 - loss: 0.0365 - val_accuracy:
0.8611 - val_loss: 0.4351
Epoch 17/100
20/20 - 19s - 939ms/step - accuracy: 0.9984 - loss: 0.0165 - val_accuracy:
0.8611 - val_loss: 0.4247
Epoch 18/100
20/20 - 19s - 933ms/step - accuracy: 0.9967 - loss: 0.0179 - val_accuracy:
0.8611 - val_loss: 0.4215
Epoch 19/100
20/20 - 19s - 929ms/step - accuracy: 1.0000 - loss: 0.0064 - val_accuracy:
0.8611 - val_loss: 0.3868
```

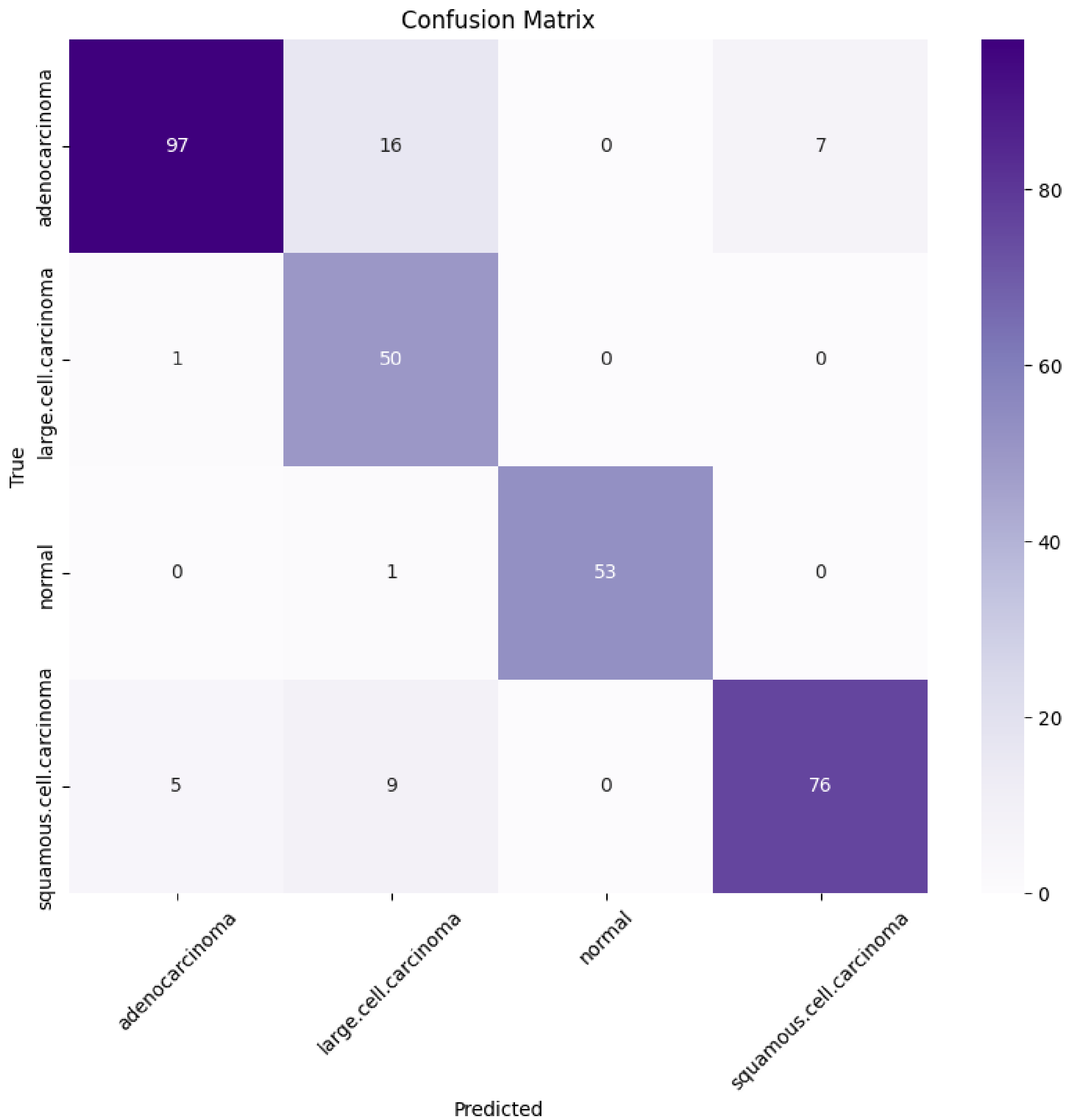
```
Epoch 20/100
20/20 - 19s - 948ms/step - accuracy: 1.0000 - loss: 0.0055 - val_accuracy:
0.8750 - val_loss: 0.4100
Epoch 21/100
20/20 - 19s - 962ms/step - accuracy: 0.9967 - loss: 0.0273 - val_accuracy:
0.8611 - val_loss: 0.4008
```

[99]:
```
plot_accuracy(history_VGG16)
```



[101]:
```
VGG16_acc, VGG16_loss = plot_confusion_matrix_and_report(VGG16_model,␣
↪test_data, class_names, './VGG16_best.weights.h5')
```

```
10/10              7s 652ms/step -
accuracy: 0.8445 - loss: 0.4848
10/10              8s 842ms/step
Classification Report:
                       precision    recall  f1-score   support

         adenocarcinoma       0.94      0.81      0.87       120
   large.cell.carcinoma       0.66      0.98      0.79        51
                 normal       1.00      0.98      0.99        54
squamous.cell.carcinoma       0.92      0.84      0.88        90

               accuracy                           0.88       315
              macro avg       0.88      0.90      0.88       315
           weighted avg       0.90      0.88      0.88       315
```



2.7 VGG19

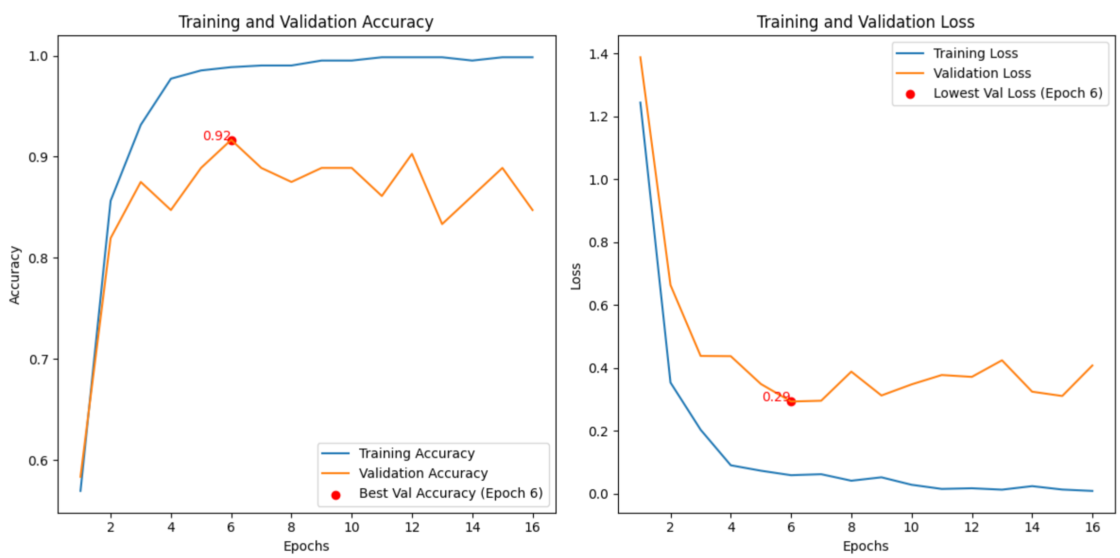
[102]:
```
VGG19_model = VGG19( include_top=False, input_shape=(224, 224, 3))
VGG19_model.trainable = False
VGG19_model = Sequential([
    VGG19_model,
    BatchNormalization(),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(4, activation='softmax')
```

```
])
VGG19_model.compile(optimizer=Adam(learning_rate),␣
↪loss='categorical_crossentropy', metrics=['accuracy'])

history_VGG19 = VGG19_model.fit(
    train_data,
    validation_data=valid_data,
    epochs = 100,
    callbacks=[early_stop, checkpoint_VGG19],
    batch_size=batch_size,
    verbose=2
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624             2s
0us/step
Epoch 1/100
20/20 - 23s - 1s/step - accuracy: 0.5693 - loss: 1.2436 - val_accuracy: 0.5833 -
val_loss: 1.3877
Epoch 2/100
20/20 - 24s - 1s/step - accuracy: 0.8564 - loss: 0.3531 - val_accuracy: 0.8194 -
val_loss: 0.6631
Epoch 3/100
20/20 - 25s - 1s/step - accuracy: 0.9315 - loss: 0.2031 - val_accuracy: 0.8750 -
val_loss: 0.4378
Epoch 4/100
20/20 - 25s - 1s/step - accuracy: 0.9772 - loss: 0.0901 - val_accuracy: 0.8472 -
val_loss: 0.4370
Epoch 5/100
20/20 - 24s - 1s/step - accuracy: 0.9853 - loss: 0.0727 - val_accuracy: 0.8889 -
val_loss: 0.3486
Epoch 6/100
20/20 - 24s - 1s/step - accuracy: 0.9886 - loss: 0.0585 - val_accuracy: 0.9167 -
val_loss: 0.2929
Epoch 7/100
20/20 - 24s - 1s/step - accuracy: 0.9902 - loss: 0.0617 - val_accuracy: 0.8889 -
val_loss: 0.2953
Epoch 8/100
20/20 - 24s - 1s/step - accuracy: 0.9902 - loss: 0.0410 - val_accuracy: 0.8750 -
val_loss: 0.3879
Epoch 9/100
20/20 - 24s - 1s/step - accuracy: 0.9951 - loss: 0.0518 - val_accuracy: 0.8889 -
val_loss: 0.3120
Epoch 10/100
20/20 - 24s - 1s/step - accuracy: 0.9951 - loss: 0.0281 - val_accuracy: 0.8889 -
val_loss: 0.3473
Epoch 11/100
```

```
20/20 - 24s - 1s/step - accuracy: 0.9984 - loss: 0.0149 - val_accuracy: 0.8611 -
val_loss: 0.3771
Epoch 12/100
20/20 - 23s - 1s/step - accuracy: 0.9984 - loss: 0.0170 - val_accuracy: 0.9028 -
val_loss: 0.3712
Epoch 13/100
20/20 - 23s - 1s/step - accuracy: 0.9984 - loss: 0.0125 - val_accuracy: 0.8333 -
val_loss: 0.4239
Epoch 14/100
20/20 - 23s - 1s/step - accuracy: 0.9951 - loss: 0.0237 - val_accuracy: 0.8611 -
val_loss: 0.3241
Epoch 15/100
20/20 - 23s - 1s/step - accuracy: 0.9984 - loss: 0.0129 - val_accuracy: 0.8889 -
val_loss: 0.3104
Epoch 16/100
20/20 - 24s - 1s/step - accuracy: 0.9984 - loss: 0.0085 - val_accuracy: 0.8472 -
val_loss: 0.4075
```

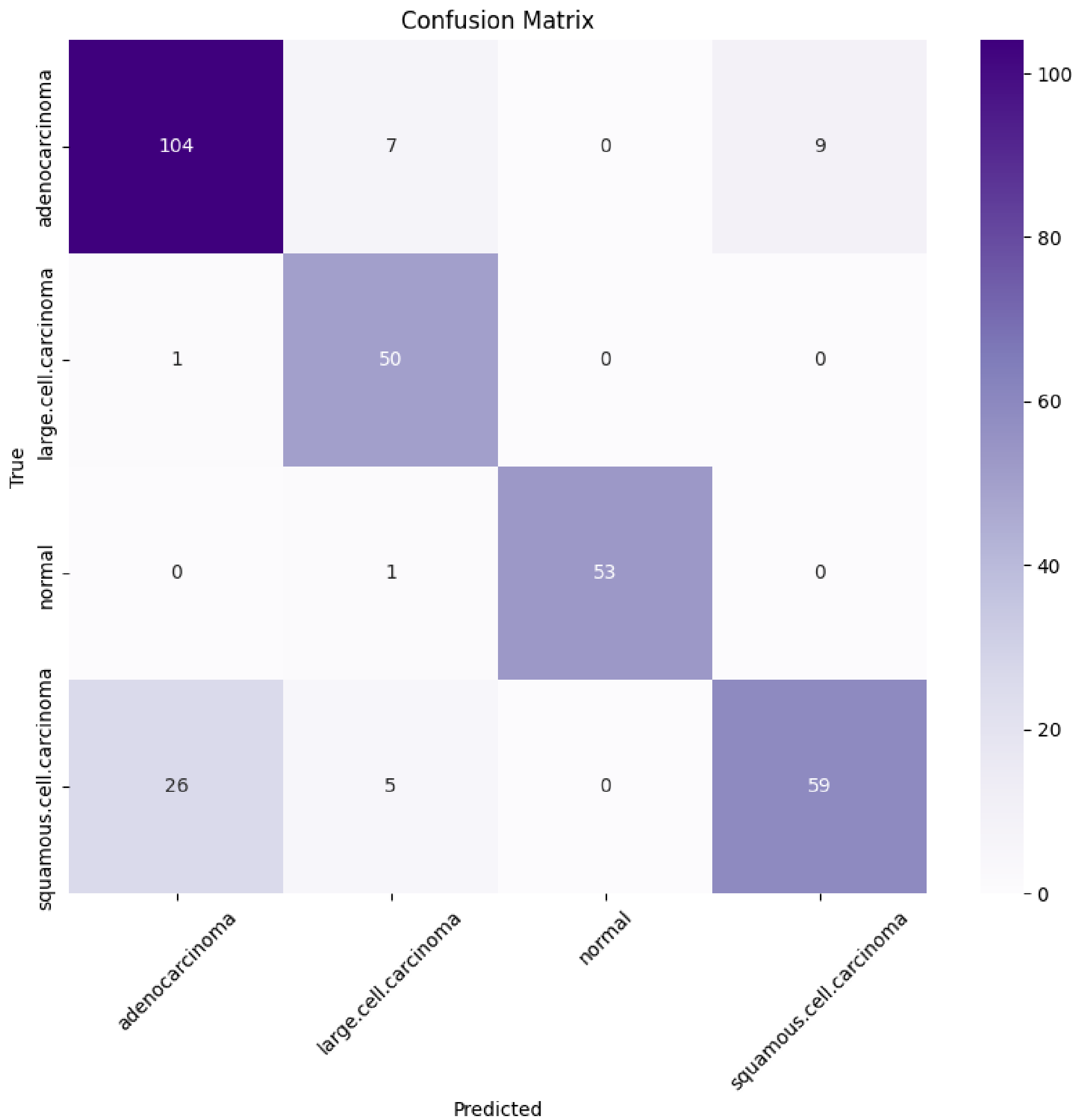
[103]:
```
plot_accuracy(history_VGG19)
```



[104]:
```
VGG19_acc, VGG19_loss = plot_confusion_matrix_and_report(VGG19_model,␣
↪test_data, class_names, './VGG19_best.weights.h5')
```

```
10/10              10s 1s/step -
accuracy: 0.8792 - loss: 0.4355
10/10              11s 1s/step
Classification Report:
                          precision    recall  f1-score   support
```

adenocarcinoma	0.79	0.87	0.83	120
large.cell.carcinoma	0.79	0.98	0.88	51
normal	1.00	0.98	0.99	54
squamous.cell.carcinoma	0.87	0.66	0.75	90
accuracy			0.84	315
macro avg	0.86	0.87	0.86	315
weighted avg	0.85	0.84	0.84	315



2.8 EfficientNetB0

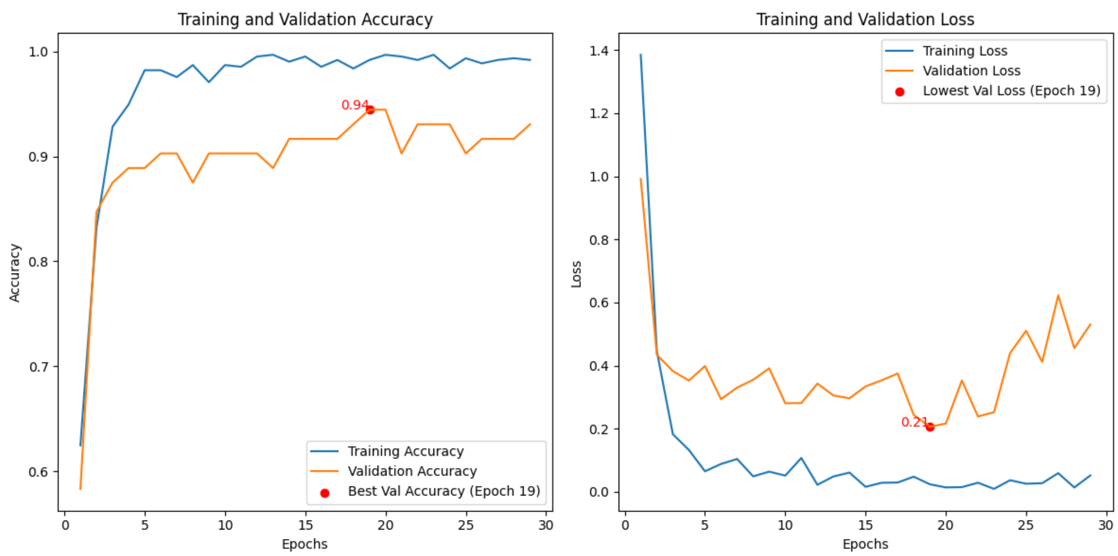
[105]:
```
EfficientNetB0_model = EfficientNetB0(include_top=False, input_shape=(224, 224,␣
↪3))
EfficientNetB0_model.trainable = False
EfficientNetB0_model = Sequential([
    EfficientNetB0_model,
    BatchNormalization(),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(4, activation='softmax')
])

EfficientNetB0_model.compile(optimizer=Adam(learning_rate=learning_rate),␣
↪loss='categorical_crossentropy', metrics=['accuracy'])

history_EfficientNetB0 = EfficientNetB0_model.fit(
    train_data,
    validation_data=valid_data,
    epochs=100,
    callbacks=[early_stop, checkpoint_EfficientNetB0],
    batch_size=batch_size,
    verbose=2
)
```

```
Downloading data from https://storage.googleapis.com/keras-
applications/efficientnetb0_notop.h5
16705208/16705208          0s
0us/step
Epoch 1/100
20/20 - 11s - 553ms/step - accuracy: 0.6248 - loss: 1.3851 - val_accuracy:
0.5833 - val_loss: 0.9918
Epoch 2/100
20/20 - 5s - 246ms/step - accuracy: 0.8320 - loss: 0.4464 - val_accuracy: 0.8472
- val_loss: 0.4346
Epoch 3/100
20/20 - 5s - 228ms/step - accuracy: 0.9282 - loss: 0.1826 - val_accuracy: 0.8750
- val_loss: 0.3829
Epoch 4/100
20/20 - 5s - 245ms/step - accuracy: 0.9494 - loss: 0.1326 - val_accuracy: 0.8889
- val_loss: 0.3526
Epoch 5/100
20/20 - 5s - 226ms/step - accuracy: 0.9821 - loss: 0.0652 - val_accuracy: 0.8889
- val_loss: 0.3987
Epoch 6/100
20/20 - 5s - 244ms/step - accuracy: 0.9821 - loss: 0.0884 - val_accuracy: 0.9028
- val_loss: 0.2937
```

```
Epoch 7/100
20/20 - 5s - 246ms/step - accuracy: 0.9755 - loss: 0.1042 - val_accuracy: 0.9028
- val_loss: 0.3304
Epoch 8/100
20/20 - 5s - 235ms/step - accuracy: 0.9869 - loss: 0.0491 - val_accuracy: 0.8750
- val_loss: 0.3551
Epoch 9/100
20/20 - 5s - 238ms/step - accuracy: 0.9706 - loss: 0.0641 - val_accuracy: 0.9028
- val_loss: 0.3915
Epoch 10/100
20/20 - 5s - 248ms/step - accuracy: 0.9869 - loss: 0.0514 - val_accuracy: 0.9028
- val_loss: 0.2808
Epoch 11/100
20/20 - 5s - 236ms/step - accuracy: 0.9853 - loss: 0.1075 - val_accuracy: 0.9028
- val_loss: 0.2814
Epoch 12/100
20/20 - 5s - 238ms/step - accuracy: 0.9951 - loss: 0.0224 - val_accuracy: 0.9028
- val_loss: 0.3431
Epoch 13/100
20/20 - 5s - 234ms/step - accuracy: 0.9967 - loss: 0.0487 - val_accuracy: 0.8889
- val_loss: 0.3056
Epoch 14/100
20/20 - 5s - 231ms/step - accuracy: 0.9902 - loss: 0.0610 - val_accuracy: 0.9167
- val_loss: 0.2966
Epoch 15/100
20/20 - 5s - 232ms/step - accuracy: 0.9951 - loss: 0.0160 - val_accuracy: 0.9167
- val_loss: 0.3341
Epoch 16/100
20/20 - 5s - 235ms/step - accuracy: 0.9853 - loss: 0.0288 - val_accuracy: 0.9167
- val_loss: 0.3534
Epoch 17/100
20/20 - 5s - 237ms/step - accuracy: 0.9918 - loss: 0.0296 - val_accuracy: 0.9167
- val_loss: 0.3751
Epoch 18/100
20/20 - 5s - 253ms/step - accuracy: 0.9837 - loss: 0.0481 - val_accuracy: 0.9306
- val_loss: 0.2438
Epoch 19/100
20/20 - 5s - 252ms/step - accuracy: 0.9918 - loss: 0.0241 - val_accuracy: 0.9444
- val_loss: 0.2065
Epoch 20/100
20/20 - 5s - 241ms/step - accuracy: 0.9967 - loss: 0.0141 - val_accuracy: 0.9444
- val_loss: 0.2164
Epoch 21/100
20/20 - 5s - 237ms/step - accuracy: 0.9951 - loss: 0.0150 - val_accuracy: 0.9028
- val_loss: 0.3532
Epoch 22/100
20/20 - 5s - 232ms/step - accuracy: 0.9918 - loss: 0.0289 - val_accuracy: 0.9306
- val_loss: 0.2389
```

```
Epoch 23/100
20/20 - 5s - 232ms/step - accuracy: 0.9967 - loss: 0.0095 - val_accuracy: 0.9306
- val_loss: 0.2524
Epoch 24/100
20/20 - 5s - 243ms/step - accuracy: 0.9837 - loss: 0.0369 - val_accuracy: 0.9306
- val_loss: 0.4399
Epoch 25/100
20/20 - 5s - 245ms/step - accuracy: 0.9935 - loss: 0.0257 - val_accuracy: 0.9028
- val_loss: 0.5106
Epoch 26/100
20/20 - 5s - 250ms/step - accuracy: 0.9886 - loss: 0.0275 - val_accuracy: 0.9167
- val_loss: 0.4119
Epoch 27/100
20/20 - 5s - 239ms/step - accuracy: 0.9918 - loss: 0.0592 - val_accuracy: 0.9167
- val_loss: 0.6228
Epoch 28/100
20/20 - 5s - 233ms/step - accuracy: 0.9935 - loss: 0.0139 - val_accuracy: 0.9167
- val_loss: 0.4557
Epoch 29/100
20/20 - 5s - 231ms/step - accuracy: 0.9918 - loss: 0.0520 - val_accuracy: 0.9306
- val_loss: 0.5303
```

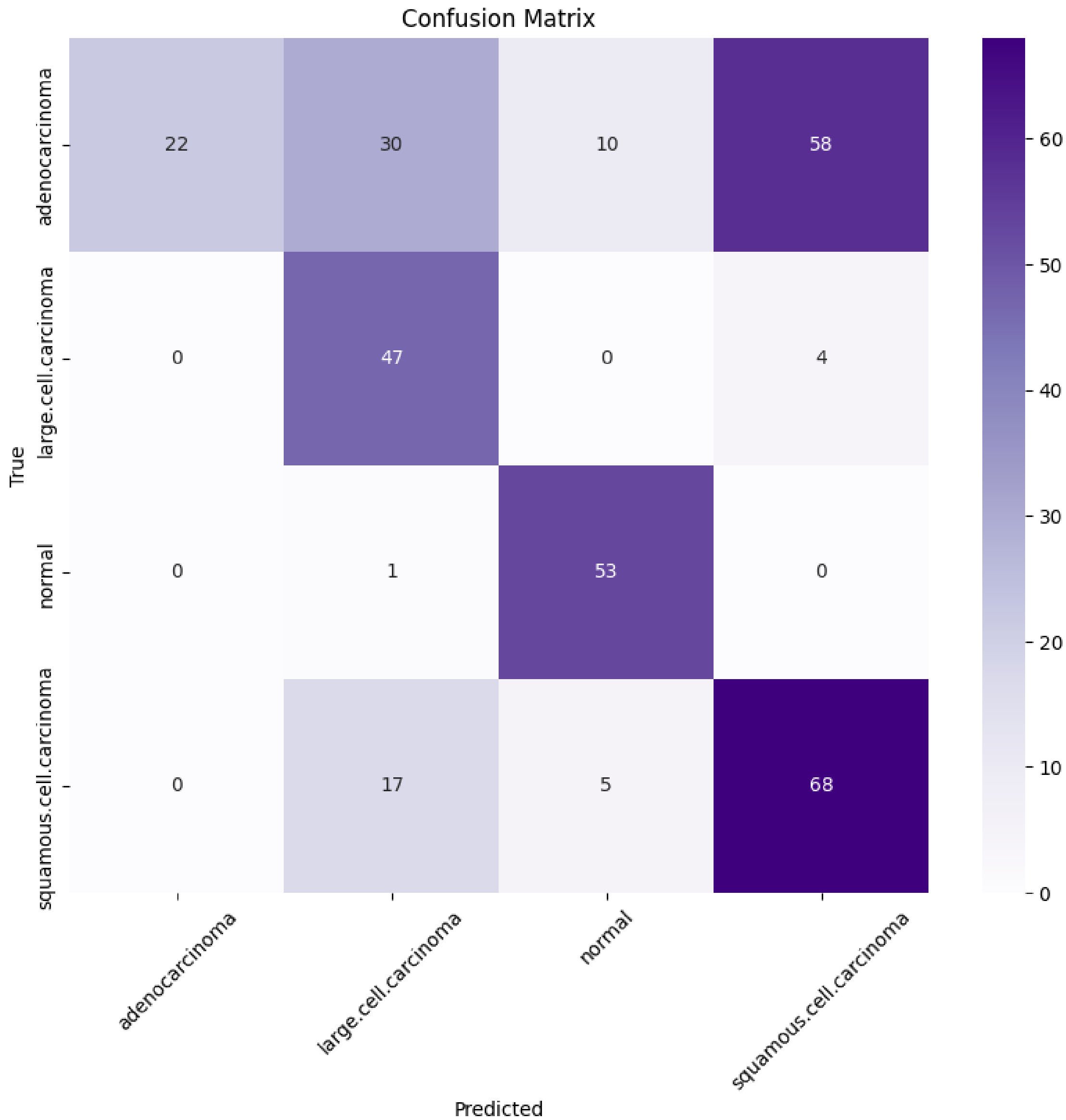
[106]:
```
plot_accuracy(history_EfficientNetB0)
```



[107]:
```
EfficientNetB0_acc, EfficientNetB0_loss =␣
↪plot_confusion_matrix_and_report(EfficientNetB0_model, test_data,␣
↪class_names, './EfficientNetB0_best.weights.h5')
```

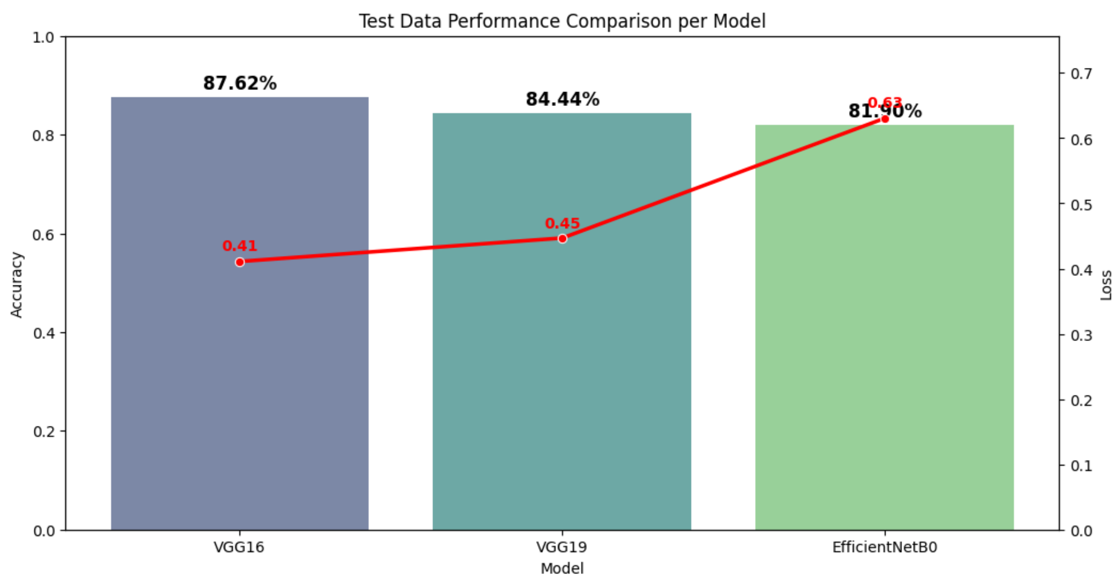
```
10/10                 2s 173ms/step -
```

```
accuracy: 0.7528 - loss: 0.8890
10/10            4s 268ms/step
Classification Report:
                         precision    recall  f1-score   support

         adenocarcinoma       1.00      0.18      0.31       120
   large.cell.carcinoma       0.49      0.92      0.64        51
                 normal       0.78      0.98      0.87        54
squamous.cell.carcinoma       0.52      0.76      0.62        90

               accuracy                           0.60       315
              macro avg       0.70      0.71      0.61       315
           weighted avg       0.74      0.60      0.55       315
```



2.9 Let's evaluate our VGGs and EfficientNetB0 models!

[119]:
```
bonus_models = ["VGG16","VGG19","EfficientNetB0"]
bonus_accs = [VGG16_acc, VGG19_acc, EfficientNetB0_acc]
bonus_loss = [VGG16_loss, VGG19_loss, EfficientNetB0_loss]

plot_comparison(bonus_models, bonus_accs, bonus_loss)
```



2.10 Initial Conclusion

2.10.1 VGG16 and VGG19 perform the best on this lung cancer image data set! 87.62% accuracy is pretty good, but let's see if we can get it up to 90% while either maintaining our 0.41 loss or lowering it as well.

Let's try: 1. Data augmentation 2. Learning rate adjustment 3. Regularization

2.11 Data augmentation for VGG16

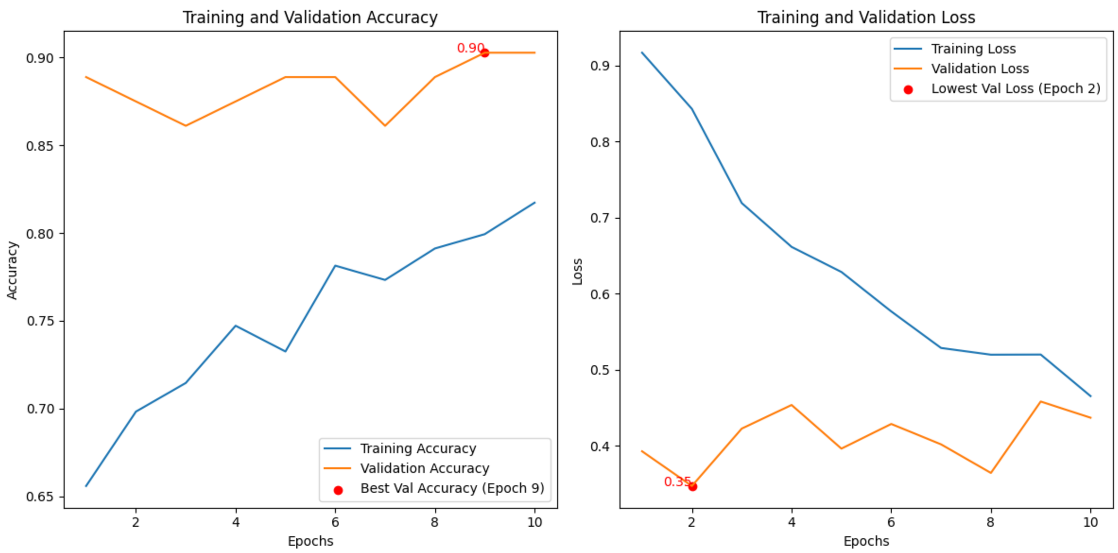
[115]:
```
# augmented checkpoint to compare to the original VGG16
checkpoint_augmented_VGG16 = ModelCheckpoint(
    'augmented_VGG16_best.weights.h5',
    monitor=monitor,
    save_best_only=True,
    save_weights_only=True,
)
```

```python
# Set up data augmentation for the training data
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

train_generator = datagen.flow_from_directory(
    './archive/Data/train',
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical'
)

valid_datagen = ImageDataGenerator()  # No augmentation for validation data
valid_generator = valid_datagen.flow_from_directory(
    './archive/Data/valid',
    shuffle=False,
    batch_size = batch_size,
    target_size = input_size,
    class_mode = "categorical"
)

# Train the model
augmented_VGG16_history = VGG16_model.fit(
    train_generator,
    validation_data=valid_generator,
    epochs=100,
    callbacks=[early_stop, checkpoint_augmented_VGG16],
    batch_size=batch_size,
    verbose=2
)
```

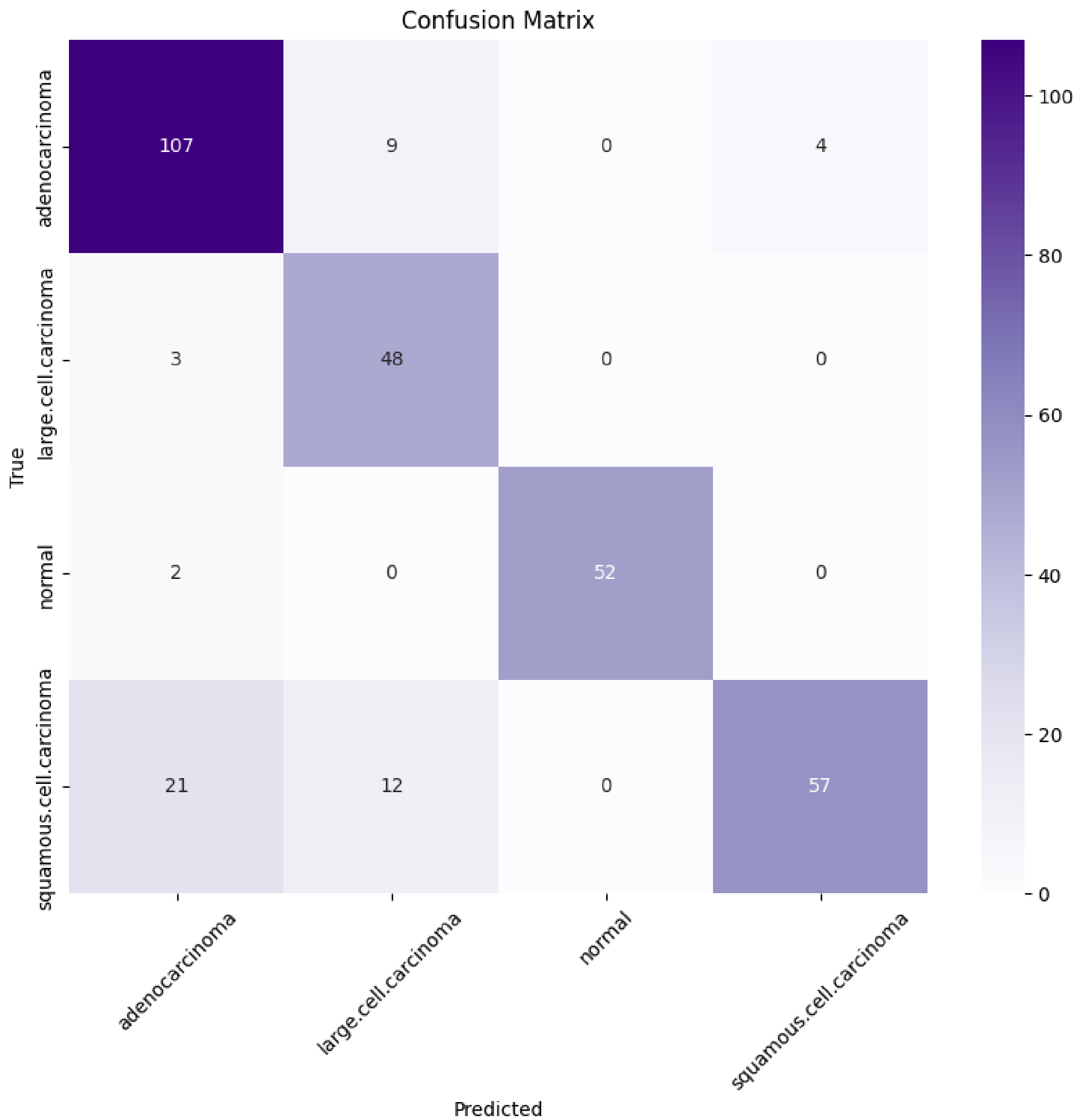
```
Found 613 images belonging to 4 classes.
Found 72 images belonging to 4 classes.
Epoch 1/100
20/20 - 18s - 878ms/step - accuracy: 0.6558 - loss: 0.9166 - val_accuracy:
0.8889 - val_loss: 0.3927
Epoch 2/100
20/20 - 20s - 986ms/step - accuracy: 0.6982 - loss: 0.8428 - val_accuracy:
0.8750 - val_loss: 0.3471
Epoch 3/100
20/20 - 19s - 971ms/step - accuracy: 0.7145 - loss: 0.7190 - val_accuracy:
0.8611 - val_loss: 0.4226
```

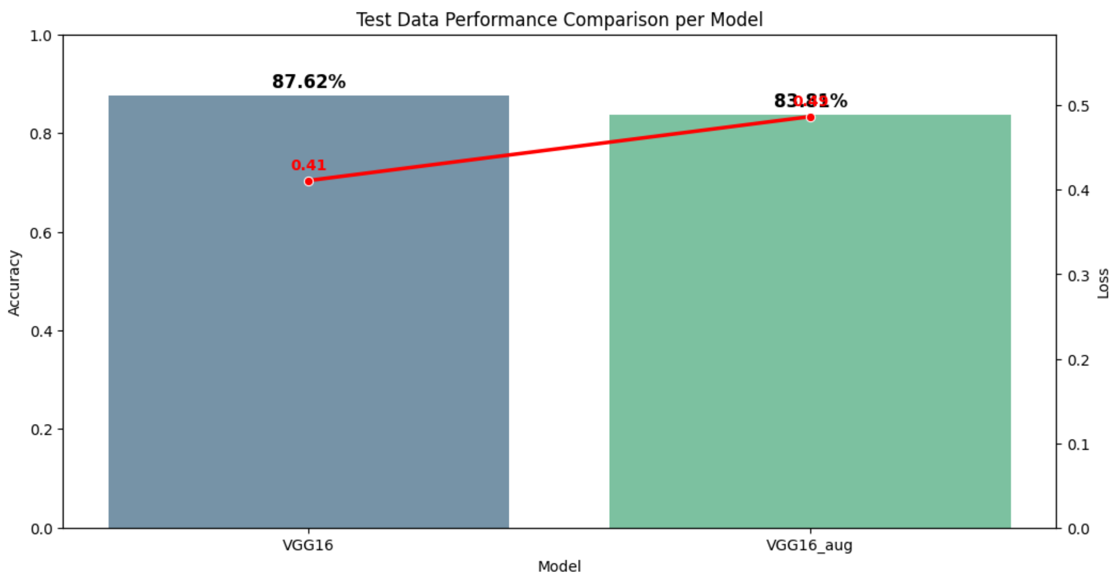
```
Epoch 4/100
20/20 - 20s - 976ms/step - accuracy: 0.7471 - loss: 0.6614 - val_accuracy:
0.8750 - val_loss: 0.4536
Epoch 5/100
20/20 - 20s - 976ms/step - accuracy: 0.7325 - loss: 0.6284 - val_accuracy:
0.8889 - val_loss: 0.3962
Epoch 6/100
20/20 - 20s - 977ms/step - accuracy: 0.7814 - loss: 0.5765 - val_accuracy:
0.8889 - val_loss: 0.4286
Epoch 7/100
20/20 - 20s - 1s/step - accuracy: 0.7732 - loss: 0.5285 - val_accuracy: 0.8611 -
val_loss: 0.4018
Epoch 8/100
20/20 - 20s - 984ms/step - accuracy: 0.7912 - loss: 0.5197 - val_accuracy:
0.8889 - val_loss: 0.3643
Epoch 9/100
20/20 - 20s - 1s/step - accuracy: 0.7993 - loss: 0.5199 - val_accuracy: 0.9028 -
val_loss: 0.4581
Epoch 10/100
20/20 - 20s - 996ms/step - accuracy: 0.8173 - loss: 0.4653 - val_accuracy:
0.9028 - val_loss: 0.4368
```

[120]:
```
# Plot accuracy for augmented VGG16
plot_accuracy(augmented_VGG16_history)

# Evaluate augmented VGG16 and plot confusion matrix
augment_VGG16_acc, augment_VGG16_loss =␣
 ↪plot_confusion_matrix_and_report(VGG16_model, test_data, class_names, './
 ↪augmented_VGG16_best.weights.h5')

# Compare models
vgg_models = ["VGG16", "VGG16_aug"]
vgg_accs = [VGG16_acc, augment_VGG16_acc]
vgg_loss = [VGG16_loss, augment_VGG16_loss]

plot_comparison(vgg_models, vgg_accs, vgg_loss)
```




```
10/10              7s 644ms/step -
accuracy: 0.8920 - loss: 0.4287
10/10              6s 640ms/step
Classification Report:
                          precision    recall  f1-score   support

         adenocarcinoma       0.80      0.89      0.85       120
   large.cell.carcinoma       0.70      0.94      0.80        51
                 normal       1.00      0.96      0.98        54
squamous.cell.carcinoma       0.93      0.63      0.75        90

               accuracy                           0.84       315
              macro avg       0.86      0.86      0.85       315
           weighted avg       0.86      0.84      0.84       315
```


Confusion Matrix
adenocarcinoma
large.cell.carcinoma
normal
squamous.cell.carcinoma
True
Predicted
107
9
0
4
3
48
0
0
2
0
52
0
21
12
0
57
0
20
40
60
80
100



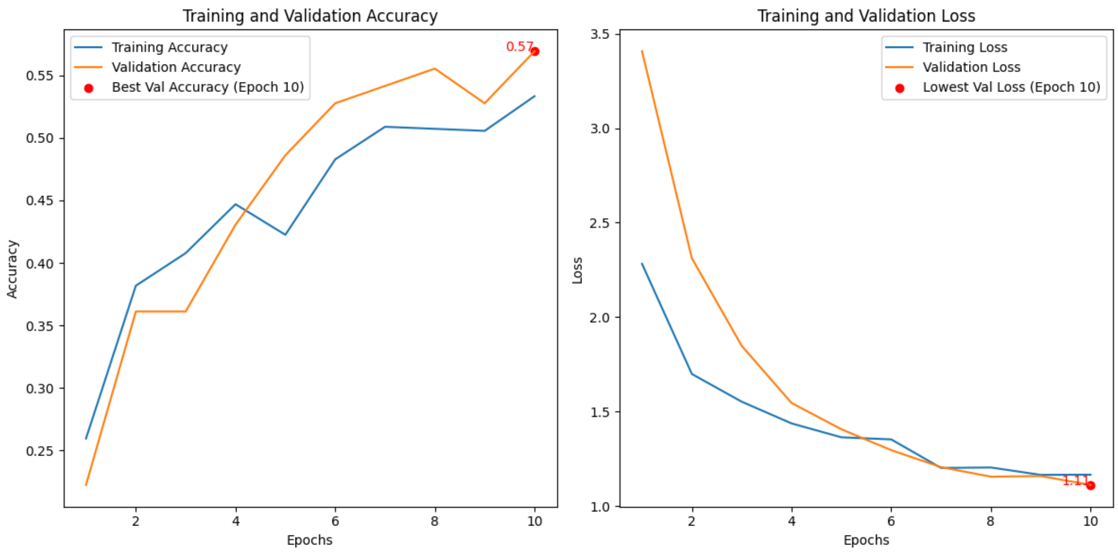
2.12 Adjust the learning rate

[143]:
```
def model_learning_rates(lr):
    checkpoint_path  = './VGG16_LR_' + str(lr) + '_best.weights.h5'
    model_checkpoint = ModelCheckpoint(checkpoint_path, monitor='val_loss',␣
↪save_best_only=True, save_weights_only=True)
    VGG16_reg_model = VGG16(include_top=False, input_shape=(224, 224, 3))
    VGG16_reg_model.trainable = False

    VGG16_reg_model = Sequential([
        VGG16_reg_model,
        BatchNormalization(),
        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(4, activation='softmax')
    ])

    VGG16_reg_model.compile(optimizer=Adam(learning_rate=lr),␣
↪loss='categorical_crossentropy', metrics=['accuracy'])

    history = VGG16_reg_model.fit(
        train_generator,
        validation_data=valid_generator,
        epochs=100,
        callbacks=[early_stop, model_checkpoint],
        batch_size=batch_size,
```

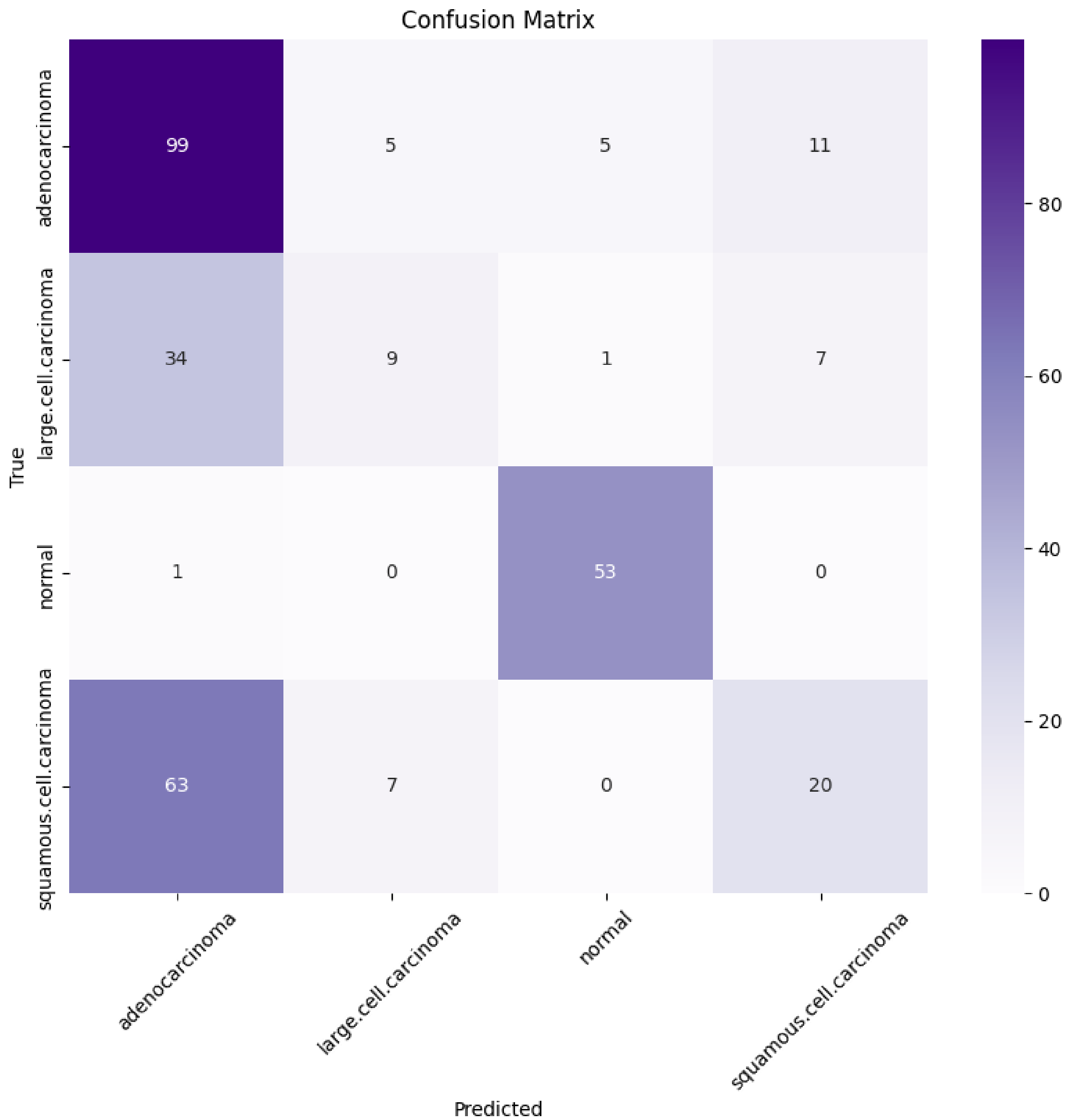
```
        verbose=2
    )
    return VGG16_reg_model, history

# Train the models with different learning rates

learning_rates = [0.00001, 0.001, 0.01] # We already have a model of 0.0001
lr_losses = [VGG16_loss]
lr_accuracies = [VGG16_acc]
lr_titles = ["VGG16_LR_0.0001"]

for lr in learning_rates:
    print(f"Training with learning rate: {lr}")
    model, history = model_learning_rates(lr)
    plot_accuracy(history)
    acc, loss = plot_confusion_matrix_and_report(model, test_data, class_names,␣
 ↪'./VGG16_LR_' + str(lr) + '_best.weights.h5')
    lr_losses.append(loss)
    lr_accuracies.append(acc)
    lr_titles.append("VGG16_LR_"+str(lr))
```

```
Training with learning rate: 1e-05
Epoch 1/100
20/20 - 17s - 865ms/step - accuracy: 0.2594 - loss: 2.2817 - val_accuracy:
0.2222 - val_loss: 3.4067
Epoch 2/100
20/20 - 20s - 1s/step - accuracy: 0.3817 - loss: 1.6989 - val_accuracy: 0.3611 -
val_loss: 2.3114
Epoch 3/100
20/20 - 20s - 1000ms/step - accuracy: 0.4078 - loss: 1.5524 - val_accuracy:
0.3611 - val_loss: 1.8479
Epoch 4/100
20/20 - 20s - 988ms/step - accuracy: 0.4470 - loss: 1.4375 - val_accuracy:
0.4306 - val_loss: 1.5457
Epoch 5/100
20/20 - 20s - 983ms/step - accuracy: 0.4225 - loss: 1.3639 - val_accuracy:
0.4861 - val_loss: 1.4065
Epoch 6/100
20/20 - 20s - 986ms/step - accuracy: 0.4829 - loss: 1.3528 - val_accuracy:
0.5278 - val_loss: 1.2960
Epoch 7/100
20/20 - 20s - 983ms/step - accuracy: 0.5090 - loss: 1.2019 - val_accuracy:
0.5417 - val_loss: 1.2061
Epoch 8/100
20/20 - 20s - 988ms/step - accuracy: 0.5073 - loss: 1.2043 - val_accuracy:
0.5556 - val_loss: 1.1556
Epoch 9/100
20/20 - 20s - 1s/step - accuracy: 0.5057 - loss: 1.1651 - val_accuracy: 0.5278 -
```

```
val_loss: 1.1591
Epoch 10/100
20/20 - 19s - 974ms/step - accuracy: 0.5334 - loss: 1.1661 - val_accuracy:
0.5694 - val_loss: 1.1119
```

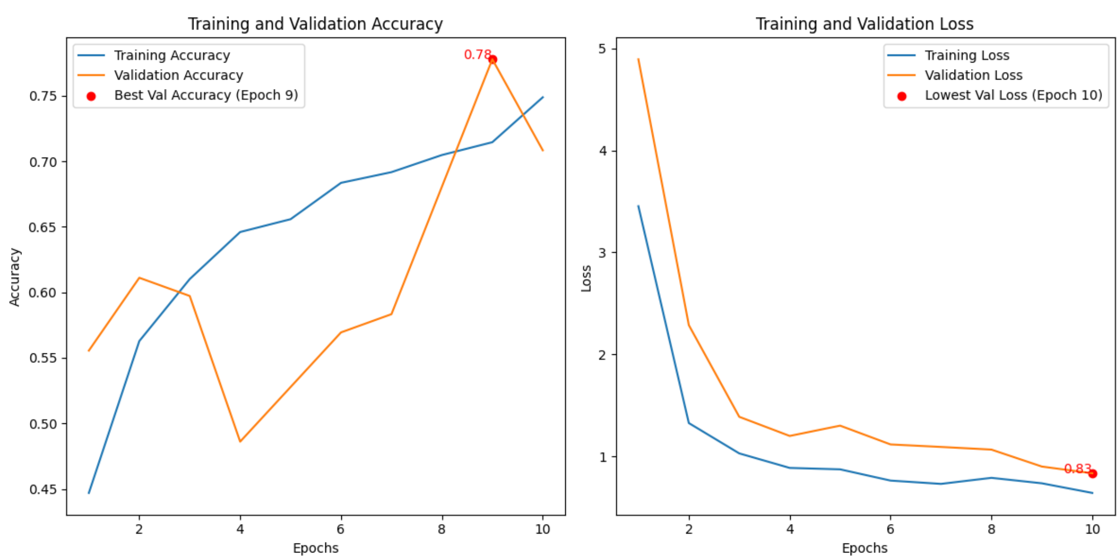



```
10/10              8s 828ms/step -
accuracy: 0.6971 - loss: 0.8279
10/10              8s 831ms/step
Classification Report:
                          precision    recall  f1-score   support

          adenocarcinoma       0.50      0.82      0.62       120
    large.cell.carcinoma       0.43      0.18      0.25        51
                  normal       0.90      0.98      0.94        54
 squamous.cell.carcinoma       0.53      0.22      0.31        90

                accuracy                           0.57       315
               macro avg       0.59      0.55      0.53       315
            weighted avg       0.57      0.57      0.53       315
```



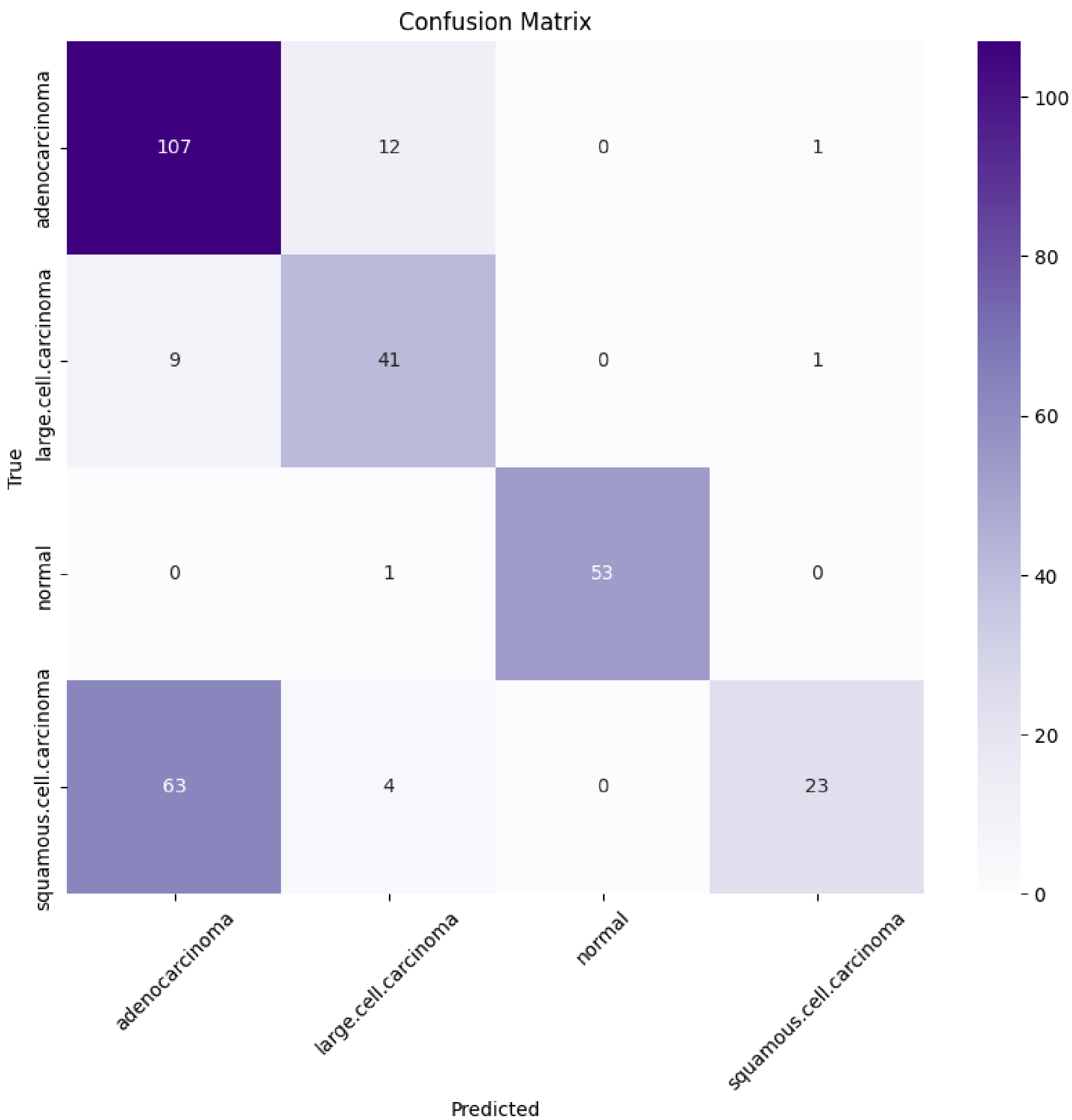

```
Training with learning rate: 0.001
Epoch 1/100
20/20 - 21s - 1s/step - accuracy: 0.4470 - loss: 3.4537 - val_accuracy: 0.5556 -
val_loss: 4.8952
Epoch 2/100
20/20 - 21s - 1s/step - accuracy: 0.5628 - loss: 1.3258 - val_accuracy: 0.6111 -
val_loss: 2.2868
Epoch 3/100
20/20 - 22s - 1s/step - accuracy: 0.6101 - loss: 1.0282 - val_accuracy: 0.5972 -
val_loss: 1.3871
Epoch 4/100
20/20 - 21s - 1s/step - accuracy: 0.6460 - loss: 0.8857 - val_accuracy: 0.4861 -
```

```
val_loss: 1.1994
Epoch 5/100
20/20 - 21s - 1s/step - accuracy: 0.6558 - loss: 0.8717 - val_accuracy: 0.5278 -
val_loss: 1.3004
Epoch 6/100
20/20 - 20s - 1s/step - accuracy: 0.6835 - loss: 0.7613 - val_accuracy: 0.5694 -
val_loss: 1.1160
Epoch 7/100
20/20 - 20s - 1s/step - accuracy: 0.6917 - loss: 0.7289 - val_accuracy: 0.5833 -
val_loss: 1.0913
Epoch 8/100
20/20 - 21s - 1s/step - accuracy: 0.7047 - loss: 0.7883 - val_accuracy: 0.6806 -
val_loss: 1.0654
Epoch 9/100
20/20 - 21s - 1s/step - accuracy: 0.7145 - loss: 0.7347 - val_accuracy: 0.7778 -
val_loss: 0.8991
Epoch 10/100
20/20 - 21s - 1s/step - accuracy: 0.7488 - loss: 0.6403 - val_accuracy: 0.7083 -
val_loss: 0.8331
```



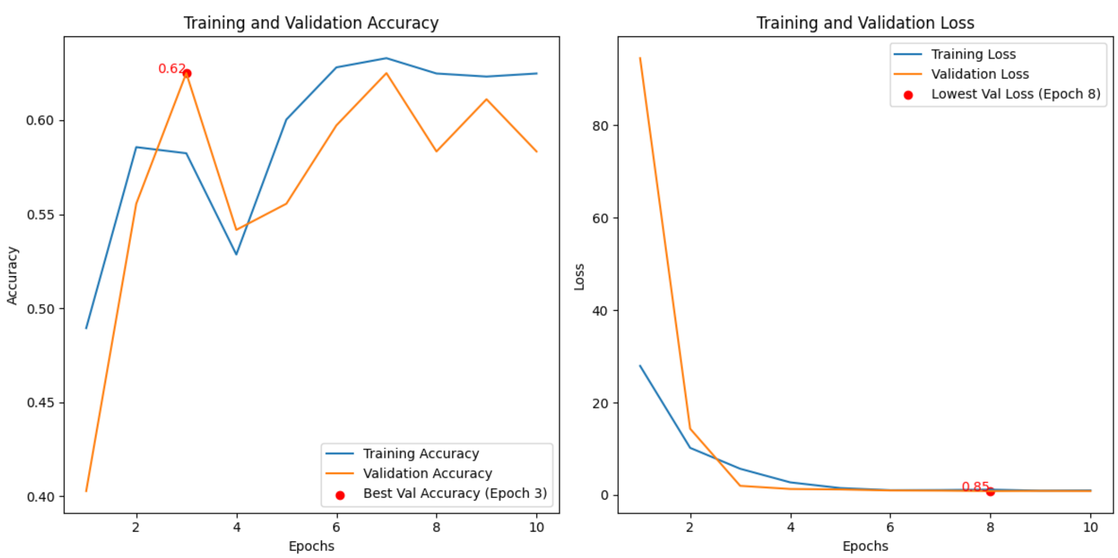

```
10/10            9s 888ms/step -
accuracy: 0.8442 - loss: 0.5711
10/10            9s 914ms/step
Classification Report:
                     precision    recall  f1-score   support

       adenocarcinoma       0.60      0.89      0.72       120
  large.cell.carcinoma       0.71      0.80      0.75        51
               normal       1.00      0.98      0.99        54
```

squamous.cell.carcinoma	0.92	0.26	0.40	90
accuracy			0.71	315
macro avg	0.81	0.73	0.71	315
weighted avg	0.78	0.71	0.68	315



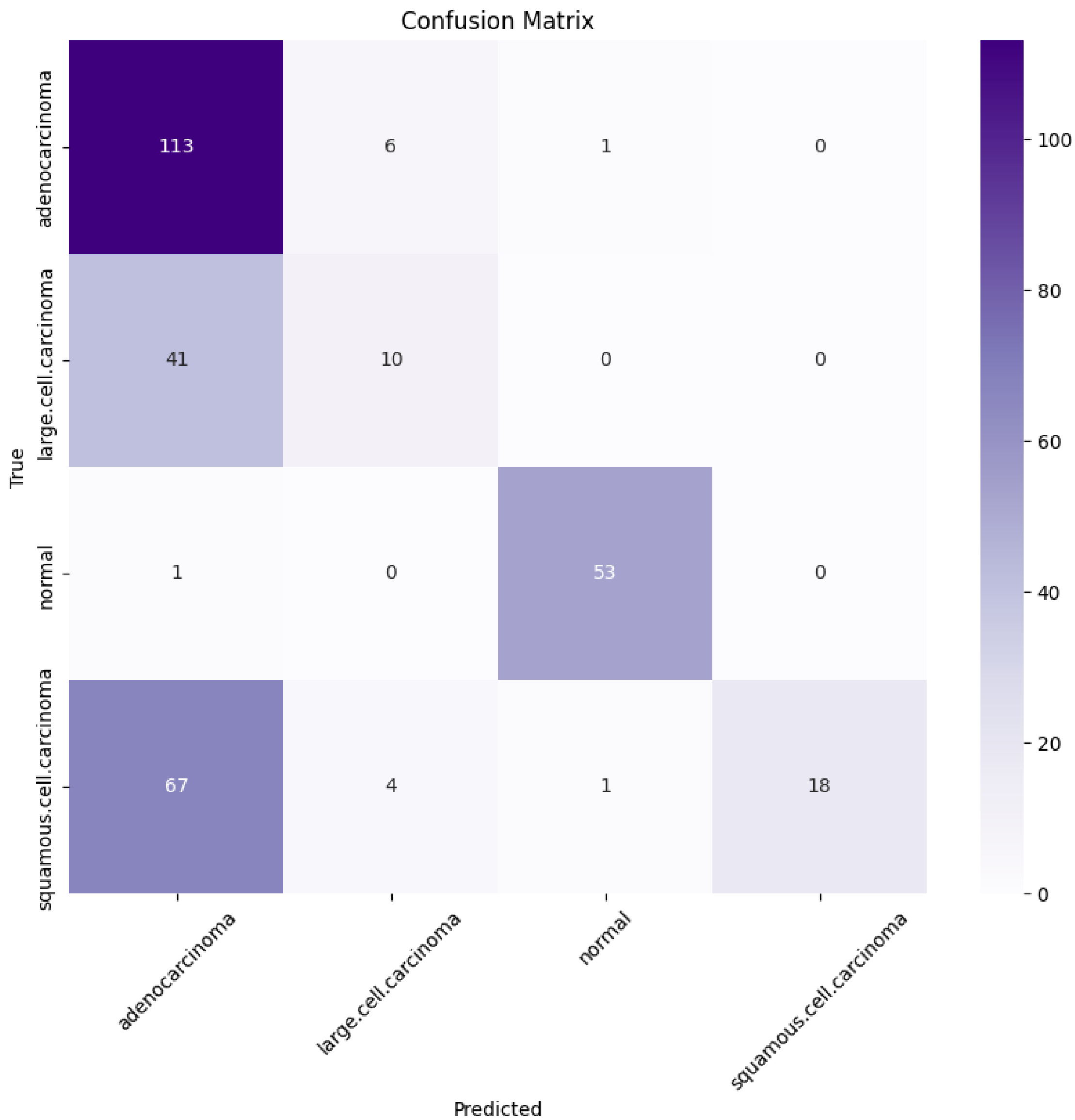

```
Training with learning rate: 0.01
Epoch 1/100
20/20 - 22s - 1s/step - accuracy: 0.4894 - loss: 27.9292 - val_accuracy: 0.4028
- val_loss: 94.5125
Epoch 2/100
```

```
20/20 - 21s - 1s/step - accuracy: 0.5856 - loss: 10.1953 - val_accuracy: 0.5556
- val_loss: 14.3409
Epoch 3/100
20/20 - 20s - 1s/step - accuracy: 0.5824 - loss: 5.6734 - val_accuracy: 0.6250 -
val_loss: 1.9773
Epoch 4/100
20/20 - 20s - 1s/step - accuracy: 0.5285 - loss: 2.7330 - val_accuracy: 0.5417 -
val_loss: 1.3005
Epoch 5/100
20/20 - 21s - 1s/step - accuracy: 0.6003 - loss: 1.4988 - val_accuracy: 0.5556 -
val_loss: 1.2025
Epoch 6/100
20/20 - 21s - 1s/step - accuracy: 0.6281 - loss: 1.0365 - val_accuracy: 0.5972 -
val_loss: 0.9790
Epoch 7/100
20/20 - 21s - 1s/step - accuracy: 0.6330 - loss: 1.0679 - val_accuracy: 0.6250 -
val_loss: 0.9423
Epoch 8/100
20/20 - 21s - 1s/step - accuracy: 0.6248 - loss: 1.1599 - val_accuracy: 0.5833 -
val_loss: 0.8513
Epoch 9/100
20/20 - 21s - 1s/step - accuracy: 0.6232 - loss: 0.9367 - val_accuracy: 0.6111 -
val_loss: 0.8893
Epoch 10/100
20/20 - 21s - 1s/step - accuracy: 0.6248 - loss: 0.9796 - val_accuracy: 0.5833 -
val_loss: 0.8605
```

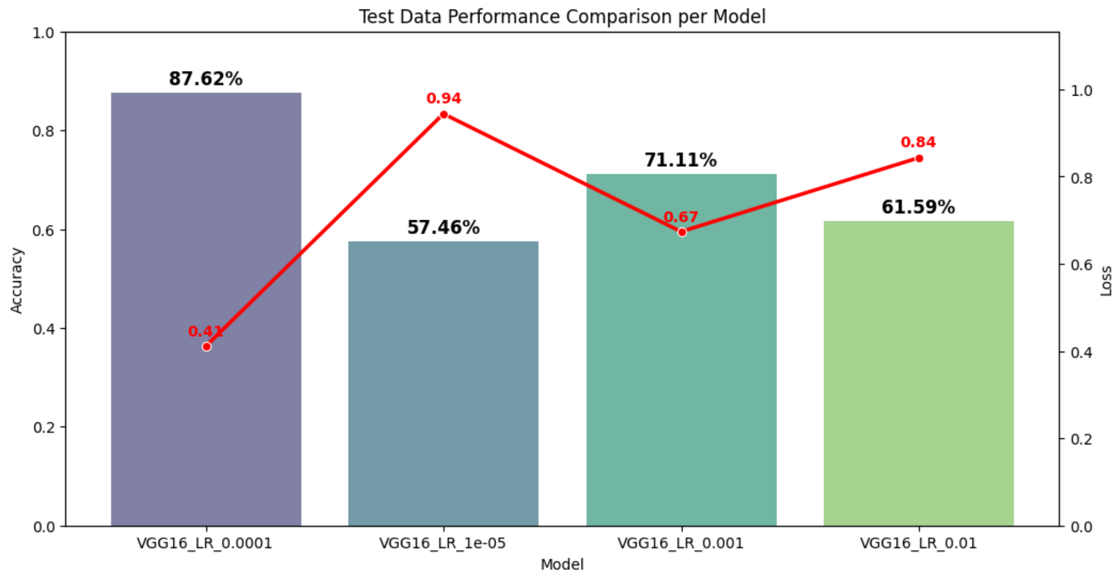



```
10/10                 9s 863ms/step -
accuracy: 0.7830 - loss: 0.7458
```

```
10/10              9s 877ms/step
Classification Report:
                          precision    recall  f1-score   support

         adenocarcinoma       0.51      0.94      0.66       120
   large.cell.carcinoma       0.50      0.20      0.28        51
                 normal       0.96      0.98      0.97        54
squamous.cell.carcinoma       1.00      0.20      0.33        90

               accuracy                           0.62       315
              macro avg       0.74      0.58      0.56       315
           weighted avg       0.73      0.62      0.56       315
```



[144]:
```
plot_comparison(lr_titles, lr_accuracies, lr_losses)
```



2.13 Regularization

Let's try adding some L2 (ridge) regularization. Regularization can be helpful when you have a complex model with lots of features like we have with our VGG16 model. L2 helps reduce overfitting by penalizing large weights.

Let's try a couple different lambdas:

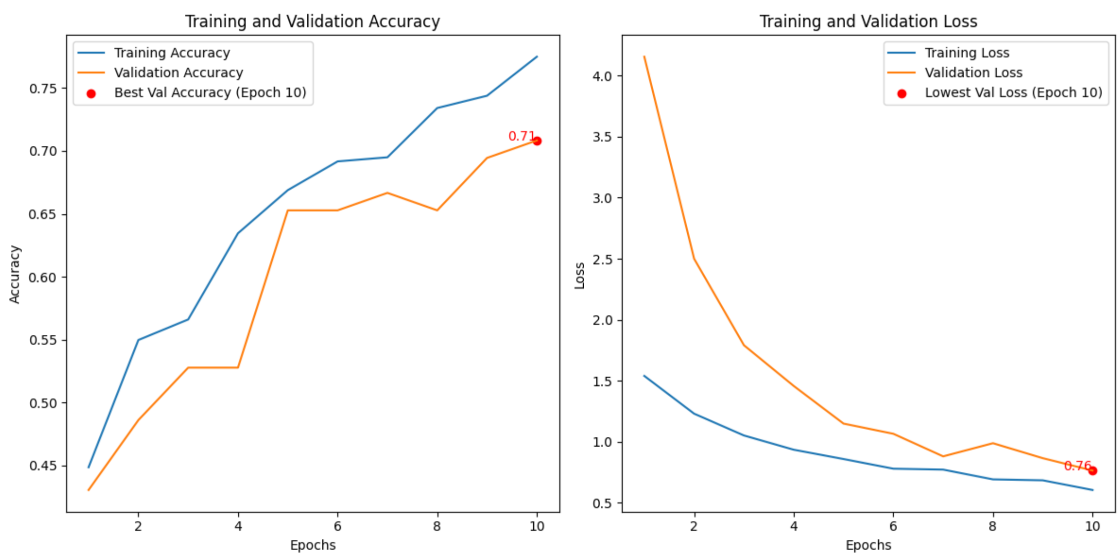
- L2 Regularization ($= 0.001$): Provides a moderate regularization effect.
- L2 Regularization (< 0.001): Weaker regularization, allowing more flexibility.
- L2 Regularization (> 0.001): Stronger regularization, reducing the model's capacity to fit noise in the training data.

[148]:
```
def model_with_lambdas(lamb):
    checkpoint_path  = './VGG16_lamda_' + str(lamb) + '_best.weights.h5'
    model_checkpoint = ModelCheckpoint(checkpoint_path, monitor='val_loss',␣
↪save_best_only=True, save_weights_only=True)
    VGG16_reg_model = VGG16(include_top=False, input_shape=(224, 224, 3))
    VGG16_reg_model.trainable = False

    VGG16_reg_model = Sequential([
        VGG16_reg_model,
        BatchNormalization(),
        Flatten(),
        Dense(256, activation='relu', kernel_regularizer=l2(lamb)),
```

```
        Dropout(0.5),
        Dense(4, activation='softmax')
    ])

    # back to our 0.0001 LR since it performed the best
    VGG16_reg_model.compile(optimizer=Adam(learning_rate=0.0001),␣
↪loss='categorical_crossentropy', metrics=['accuracy'])

    history = VGG16_reg_model.fit(
        train_generator,
        validation_data=valid_generator,
        epochs=100,
        callbacks=[early_stop, model_checkpoint],
        batch_size=batch_size,
        verbose=2
    )
    return VGG16_reg_model, history

# Train the models with different lambdas
l2_lambdas = [0.0001, 0.001, 0.01]
lambda_losses = []
lambda_accuracies = []
lambda_titles = []

for lam in l2_lambdas:
    print(f"Training with l2 reg: {lam}")
    model, history = model_with_lambdas(lam)
    plot_accuracy(history)
    acc, loss = plot_confusion_matrix_and_report(model, test_data, class_names,␣
↪'./VGG16_lamda_' + str(lam) + '_best.weights.h5')
    lambda_losses.append(loss)
    lambda_accuracies.append(acc)
    lambda_titles.append("VGG16_LR_"+str(lam))
```

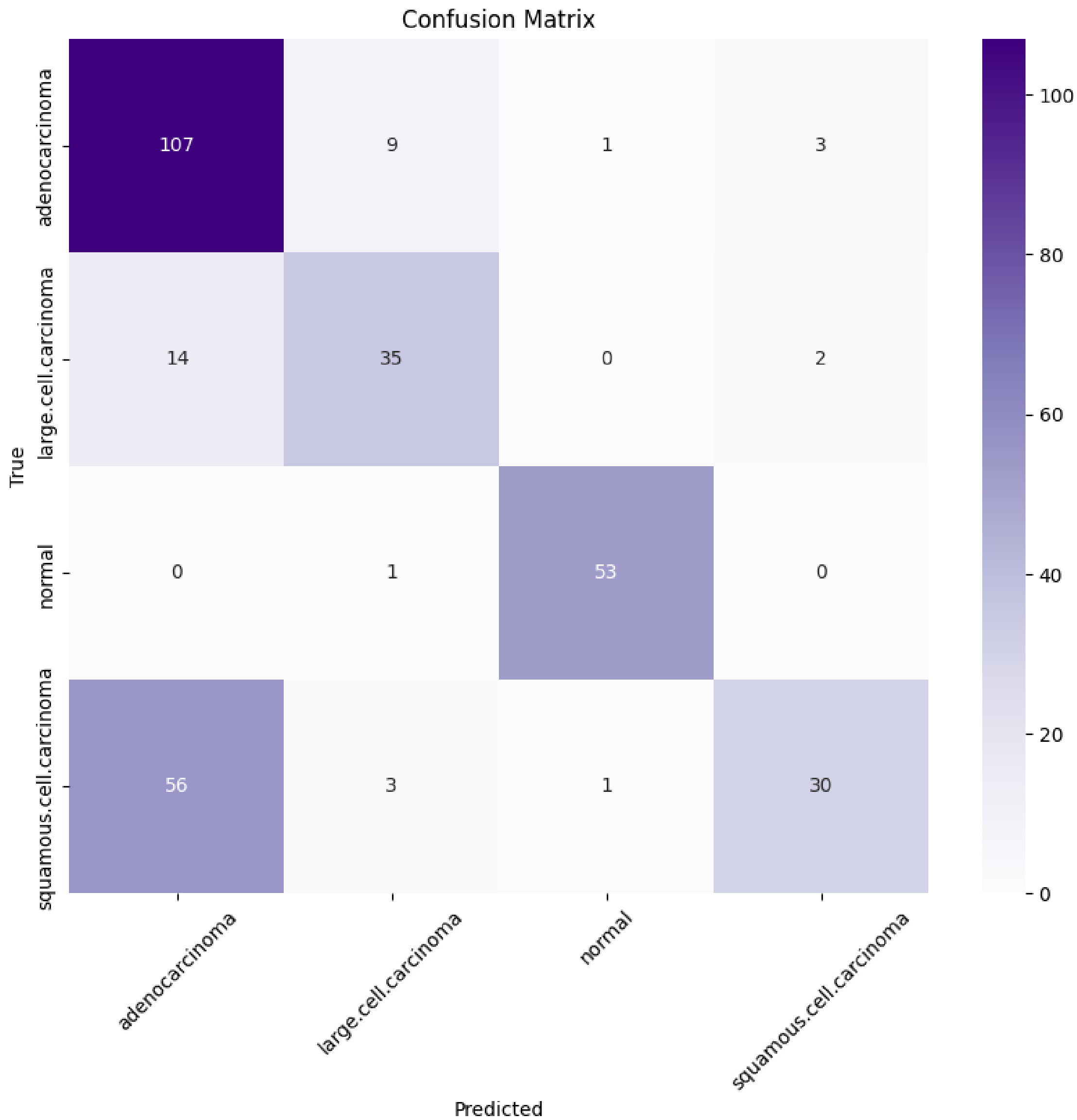
```
Training with l2 reg: 0.0001
Epoch 1/100
20/20 - 19s - 963ms/step - accuracy: 0.4486 - loss: 1.5397 - val_accuracy:
0.4306 - val_loss: 4.1556
Epoch 2/100
20/20 - 20s - 1s/step - accuracy: 0.5498 - loss: 1.2304 - val_accuracy: 0.4861 -
val_loss: 2.5002
Epoch 3/100
20/20 - 20s - 997ms/step - accuracy: 0.5661 - loss: 1.0517 - val_accuracy:
0.5278 - val_loss: 1.7914
Epoch 4/100
20/20 - 20s - 988ms/step - accuracy: 0.6346 - loss: 0.9350 - val_accuracy:
0.5278 - val_loss: 1.4576
```

```
Epoch 5/100
20/20 - 20s - 992ms/step - accuracy: 0.6688 - loss: 0.8586 - val_accuracy:
0.6528 - val_loss: 1.1491
Epoch 6/100
20/20 - 20s - 991ms/step - accuracy: 0.6917 - loss: 0.7795 - val_accuracy:
0.6528 - val_loss: 1.0656
Epoch 7/100
20/20 - 20s - 990ms/step - accuracy: 0.6949 - loss: 0.7720 - val_accuracy:
0.6667 - val_loss: 0.8806
Epoch 8/100
20/20 - 20s - 988ms/step - accuracy: 0.7341 - loss: 0.6919 - val_accuracy:
0.6528 - val_loss: 0.9882
Epoch 9/100
20/20 - 20s - 1s/step - accuracy: 0.7439 - loss: 0.6841 - val_accuracy: 0.6944 -
val_loss: 0.8657
Epoch 10/100
20/20 - 20s - 983ms/step - accuracy: 0.7749 - loss: 0.6049 - val_accuracy:
0.7083 - val_loss: 0.7641
```



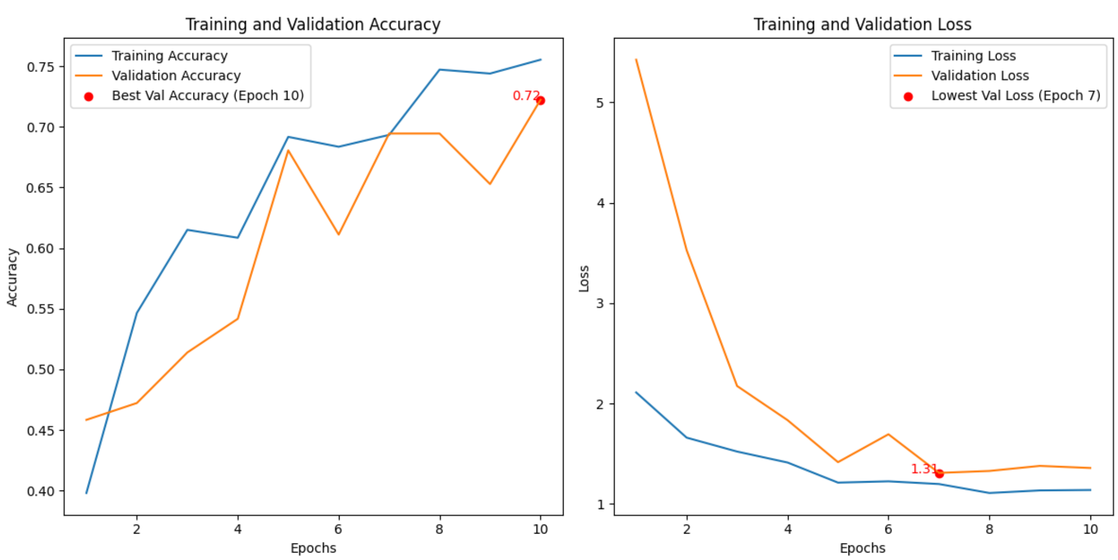

```
10/10              8s 829ms/step -
accuracy: 0.8390 - loss: 0.5216
10/10              8s 833ms/step
Classification Report:
                          precision    recall  f1-score   support

         adenocarcinoma       0.60      0.89      0.72       120
   large.cell.carcinoma       0.73      0.69      0.71        51
                 normal       0.96      0.98      0.97        54
squamous.cell.carcinoma       0.86      0.33      0.48        90
```

accuracy			0.71	315
macro avg	0.79	0.72	0.72	315
weighted avg	0.76	0.71	0.69	315




```
Training with l2 reg: 0.001
Epoch 1/100
20/20 - 21s - 1s/step - accuracy: 0.3980 - loss: 2.1102 - val_accuracy: 0.4583 -
val_loss: 5.4237
Epoch 2/100
20/20 - 20s - 992ms/step - accuracy: 0.5465 - loss: 1.6602 - val_accuracy:
```

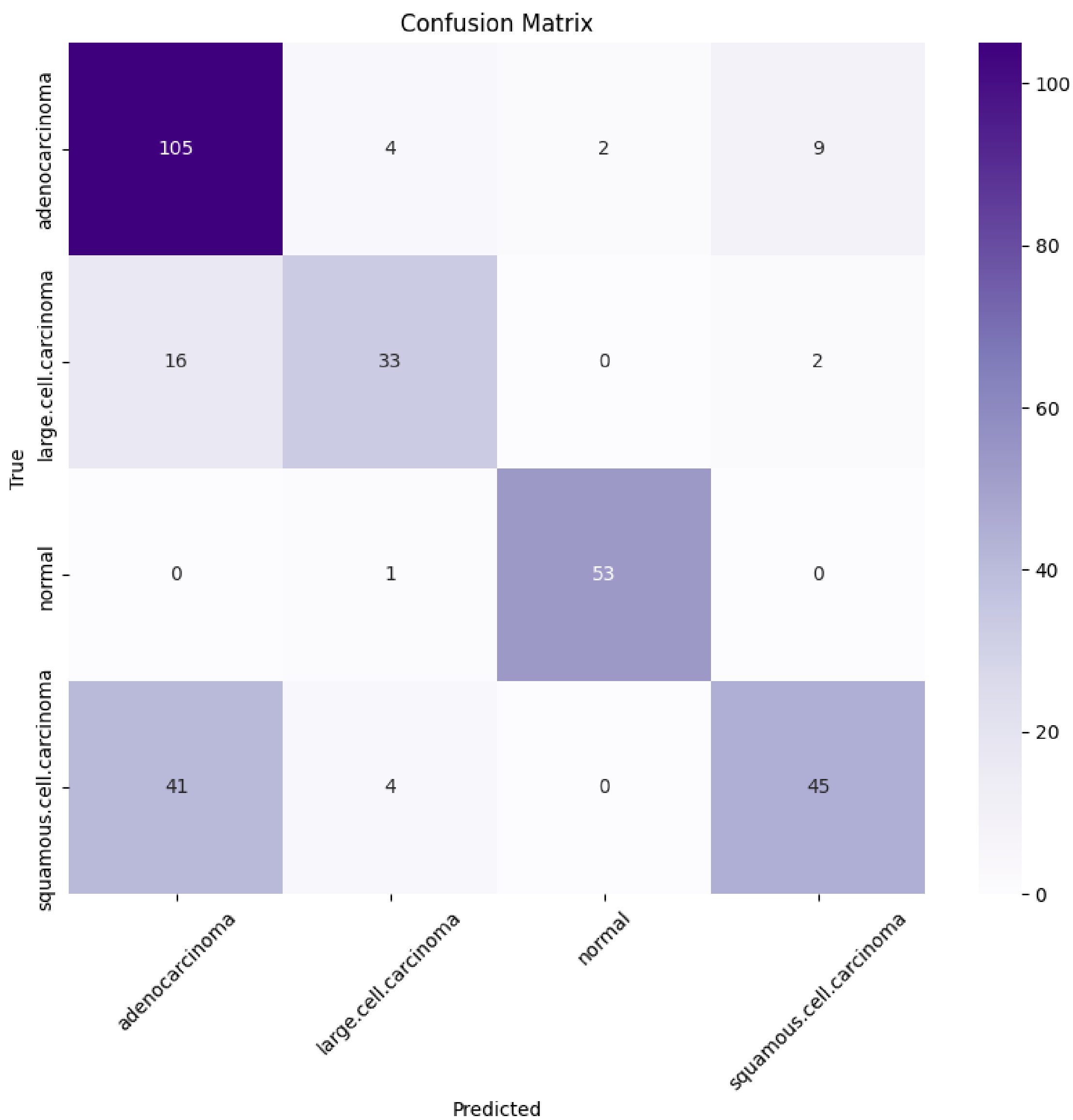
```
0.4722 - val_loss: 3.5271
Epoch 3/100
20/20 - 20s - 979ms/step - accuracy: 0.6150 - loss: 1.5216 - val_accuracy:
0.5139 - val_loss: 2.1746
Epoch 4/100
20/20 - 20s - 988ms/step - accuracy: 0.6085 - loss: 1.4125 - val_accuracy:
0.5417 - val_loss: 1.8344
Epoch 5/100
20/20 - 21s - 1s/step - accuracy: 0.6917 - loss: 1.2123 - val_accuracy: 0.6806 -
val_loss: 1.4161
Epoch 6/100
20/20 - 20s - 999ms/step - accuracy: 0.6835 - loss: 1.2252 - val_accuracy:
0.6111 - val_loss: 1.6937
Epoch 7/100
20/20 - 21s - 1s/step - accuracy: 0.6933 - loss: 1.1983 - val_accuracy: 0.6944 -
val_loss: 1.3090
Epoch 8/100
20/20 - 21s - 1s/step - accuracy: 0.7471 - loss: 1.1090 - val_accuracy: 0.6944 -
val_loss: 1.3283
Epoch 9/100
20/20 - 21s - 1s/step - accuracy: 0.7439 - loss: 1.1347 - val_accuracy: 0.6528 -
val_loss: 1.3788
Epoch 10/100
20/20 - 20s - 1s/step - accuracy: 0.7553 - loss: 1.1389 - val_accuracy: 0.7222 -
val_loss: 1.3580
```



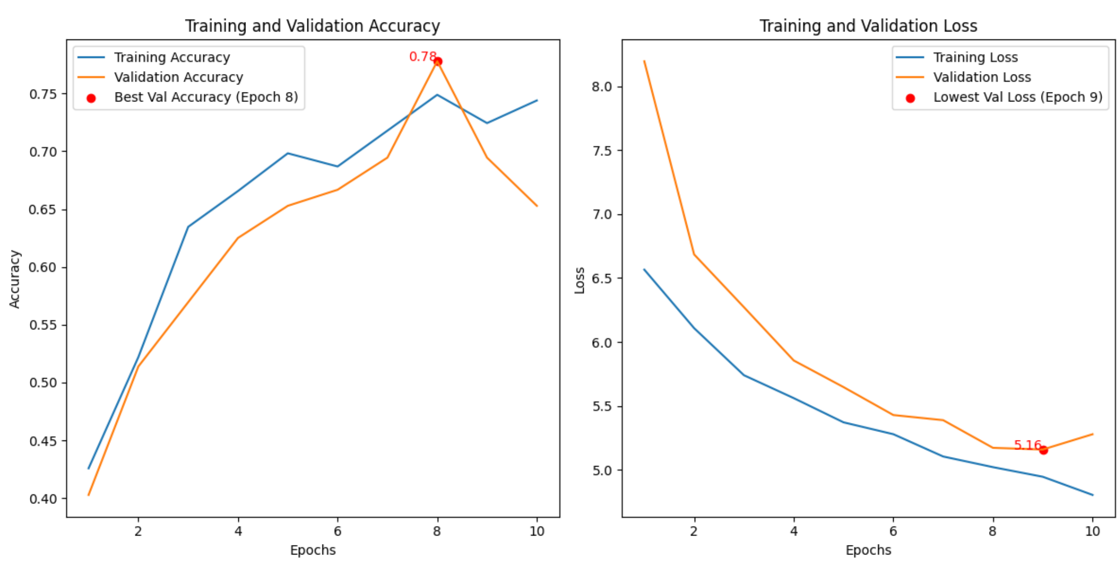

```
10/10            9s 895ms/step -
accuracy: 0.8179 - loss: 1.0498
10/10            9s 902ms/step
```

Classification Report:

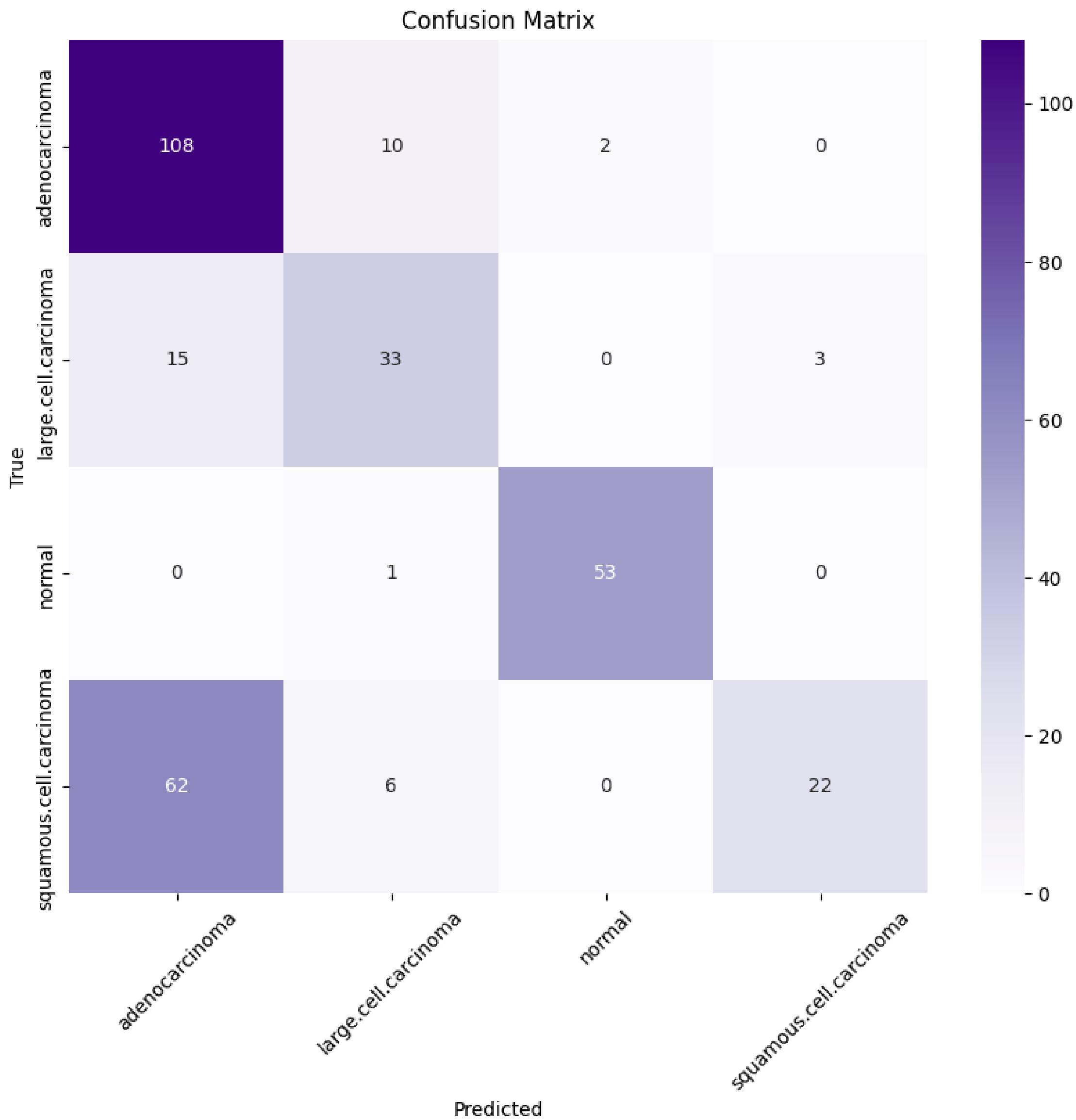
	precision	recall	f1-score	support
adenocarcinoma	0.65	0.88	0.74	120
large.cell.carcinoma	0.79	0.65	0.71	51
normal	0.96	0.98	0.97	54
squamous.cell.carcinoma	0.80	0.50	0.62	90
accuracy			0.75	315
macro avg	0.80	0.75	0.76	315
weighted avg	0.77	0.75	0.74	315



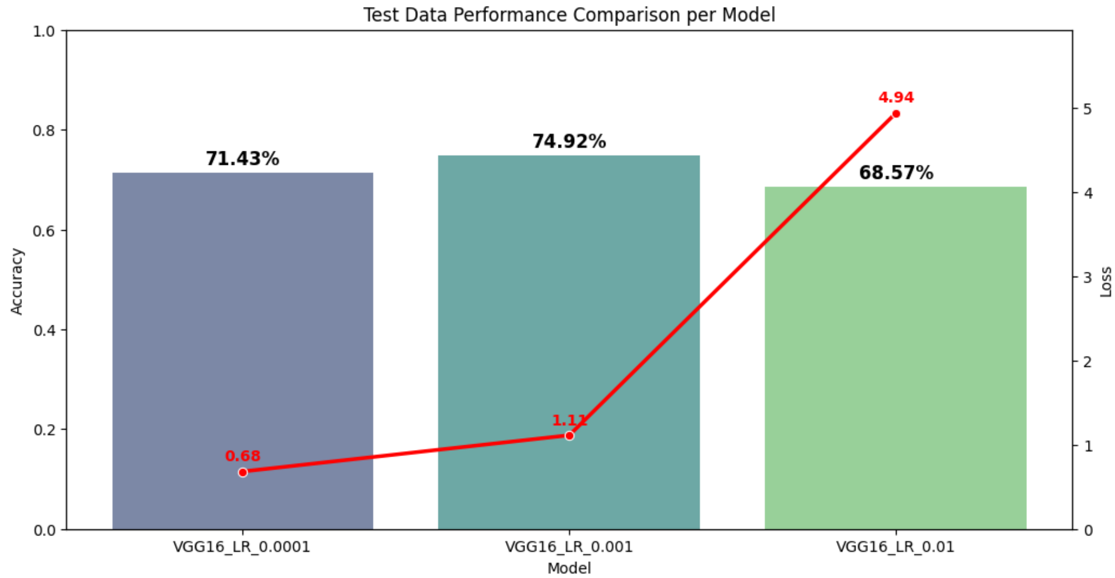
```
Training with l2 reg: 0.01
Epoch 1/100
20/20 - 23s - 1s/step - accuracy: 0.4258 - loss: 6.5659 - val_accuracy: 0.4028 -
val_loss: 8.1972
Epoch 2/100
20/20 - 20s - 1s/step - accuracy: 0.5220 - loss: 6.1080 - val_accuracy: 0.5139 -
val_loss: 6.6850
Epoch 3/100
20/20 - 20s - 1s/step - accuracy: 0.6346 - loss: 5.7394 - val_accuracy: 0.5694 -
val_loss: 6.2721
Epoch 4/100
20/20 - 20s - 1s/step - accuracy: 0.6656 - loss: 5.5601 - val_accuracy: 0.6250 -
val_loss: 5.8547
Epoch 5/100
20/20 - 21s - 1s/step - accuracy: 0.6982 - loss: 5.3705 - val_accuracy: 0.6528 -
val_loss: 5.6463
Epoch 6/100
20/20 - 22s - 1s/step - accuracy: 0.6868 - loss: 5.2777 - val_accuracy: 0.6667 -
val_loss: 5.4277
Epoch 7/100
20/20 - 21s - 1s/step - accuracy: 0.7178 - loss: 5.1031 - val_accuracy: 0.6944 -
val_loss: 5.3875
Epoch 8/100
20/20 - 20s - 1s/step - accuracy: 0.7488 - loss: 5.0198 - val_accuracy: 0.7778 -
val_loss: 5.1709
Epoch 9/100
20/20 - 20s - 999ms/step - accuracy: 0.7243 - loss: 4.9444 - val_accuracy:
0.6944 - val_loss: 5.1578
Epoch 10/100
20/20 - 20s - 994ms/step - accuracy: 0.7439 - loss: 4.8023 - val_accuracy:
0.6528 - val_loss: 5.2775
```



```
10/10             8s 823ms/step -
accuracy: 0.8083 - loss: 4.7617
10/10             8s 834ms/step
Classification Report:
                         precision    recall  f1-score   support

         adenocarcinoma       0.58      0.90      0.71       120
   large.cell.carcinoma       0.66      0.65      0.65        51
                 normal       0.96      0.98      0.97        54
squamous.cell.carcinoma       0.88      0.24      0.38        90

               accuracy                           0.69       315
              macro avg       0.77      0.69      0.68       315
           weighted avg       0.75      0.69      0.65       315
```



[151]:
```
plot_comparison(["VGG16_LR_"+str(0.0001),"VGG16_LR_"+str(0.
↪001),"VGG16_LR_"+str(0.01)], lambda_accuracies, lambda_losses)
```



3 Conclusion

The hyperparameter investigation led to worse performance/overfitting. Therefore... ## Out of the box VGG16 is the best model to use for this lung cancer set!

[ ]:

```

```