

Assignment 6

CS4172 Machine Learning Lab

Name: Abhiroop Mukherjee

Enrolment Number: 510519109

Task 1

Download and extract the flower image dataset from <https://www.kaggle.com/alxmamaev/flowers-recognition>.

Task 2

The dataset contains five classes of flower images of variable sizenamely chamomile, tulip, rose, sunflower, dandelion. Resize all images to 80*80 pixels and convert all color images to grey images.

Task 3

Randomly shuffle all images to create training, test set with ratio of 90:10, respectively. (Reduce the training size by 1/ 5 if computation resources are limited).

```
In [ ]: import tensorflow as tf
import random

import os
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3" #ignore CUDA messages

DATASET_DIRECTORY_PATH = '../ML_DRIVE/Assign_6/flowers/'
seed = random.randint(0,100)

train_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    directory= DATASET_DIRECTORY_PATH,
    labels="inferred",
    label_mode="categorical", #output of model will be softmax categorical
    color_mode="grayscale",
    batch_size=32,
    image_size=(80,80),
    shuffle=True,
    seed=seed, #same seed for both dataset so that no overlap happens
    validation_split=0.1, #90:10 split
    subset='training',
    crop_to_aspect_ratio=False,
)

val_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    directory= DATASET_DIRECTORY_PATH,
    labels="inferred",
    label_mode="categorical", #output of model will be softmax categorical
    color_mode="grayscale",
    batch_size=32,
    image_size=(80,80),
    shuffle=True,
```

```
    seed=seed,
    validation_split=0.1, #90:10 split
    subset='validation',
    crop_to_aspect_ratio=False,
)
```

```
Found 4317 files belonging to 5 classes.
Using 3886 files for training.
Found 4317 files belonging to 5 classes.
Using 431 files for validation.
```

```
In [ ]: train_dataset
```

```
Out[ ]: <BatchDataset element_spec=(TensorSpec(shape=(None, 80, 80, 1), dtype=tf.float32, name=None), TensorSpec(shape=(None, 5), dtype=tf.float32, name=None))>
```

Task 4

Train a Convolutional neural network with max pooling and a fully connected layer at the top, to classify the flower images. Now run the network by changing the following hyper-parameters:

```
In [ ]: input_shape = (80, 80, 1)
num_class = 5
```

```
In [ ]: from tensorflow.keras import Sequential
from tensorflow.keras.layers import \
    Conv2D, Dense, Input, MaxPool2D, Flatten, \
    Dropout, AvgPool2D, LeakyReLU, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.losses import CategoricalCrossentropy
import matplotlib.pyplot as plt
from tensorflow.keras.metrics import Precision, Recall
from tensorflow.math import confusion_matrix
import numpy as np
import seaborn as sns

LRELU_ALPHA = 0.01

def _plot_history(
    history: 'tf.keras.callbacks.History',
    conv_kernels: 'list[tuple[int, int]]',
    conv_filters: 'list[int]',
    activation: 'str',
    pool: 'str',
    num_fc_layers: 'int'
):
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Training')
    plt.plot(history.history['val_loss'], label='Validation')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend()
    plt.title(f'Loss vs epoch')

    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Training')
    plt.plot(history.history['val_accuracy'], label='Validation')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
```

```

plt.legend()
plt.title('Accuracy vs epoch')

plt.suptitle(
    f'filters {conv_filters}; kernels {conv_kernels}; {pool}_pool; {activation}; dense layers after flatten= {num_fc_layers}')
plt.show()

def _plot_confusion_matrix(y_val: 'list[int]',
                           y_pred: 'list[int]'):
    matrix = confusion_matrix(y_val, y_pred)
    fig = plt.figure(figsize=(12, 5))
    sns.heatmap(
        matrix,
        xticklabels=range(1, num_class + 1),
        yticklabels=range(1, num_class + 1),
        linewidth=0.5,
        cmap='coolwarm',
        annot=True,
        cbar=True)
    plt.title('Confusion Matrix for the above model')
    plt.ylabel('Actual Value')
    plt.xlabel('Predicted Value')
    plt.show()

def train_model(
    conv_kernels: 'list[tuple[int, int]]',
    conv_filters: 'list[int]',
    activation: 'str',
    pool: 'str',
    drop_rate: 'float',
    num_fc_layers: 'int',
    train_dataset: "tf.data.Dataset",
    val_dataset: "tf.data.Dataset",
    add_batch_norm=False,
    fc_layer_size=64,
    num_epochs=100,
    extra_conv_layers=0,
    give_model=False
):
    model = Sequential()
    model.add(Input(shape=input_shape))

    for filtr, kernel in zip(conv_filters, conv_kernels):
        if activation == 'lrelu':
            model.add(Conv2D(
                filters=filtr,
                kernel_size=kernel,
                activation=LeakyReLU(alpha=LRELU_APLHA)
            ))
        else:
            model.add(Conv2D(
                filters=filtr,
                kernel_size=kernel,
                activation=activation
            ))

    if pool.lower() == 'max':
        model.add(MaxPool2D())

```

```

    elif pool.lower() == 'avg':
        model.add(AvgPool2D())
    else:
        raise Exception('argument pool is undefined')

    if add_batch_norm:
        model.add(BatchNormalization())

    if drop_rate > 0:
        model.add(Dropout(rate=drop_rate))

# add extra_conv_layers

# also padding "same" cause image dimension is reducing to 0 otherwise
for i in range(0, extra_conv_layers):
    conv_filters.append(conv_filters[-1]*2)
    conv_kernels.append(conv_kernels[-1])

    if activation == 'lrelu':
        model.add(Conv2D(
            filters=conv_filters[-1],
            kernel_size=conv_kernels[-1],
            activation=LeakyReLU(alpha=LRELU_APLHA),
            padding='same'
        ))
    else:
        model.add(Conv2D(
            filters=conv_filters[-1],
            kernel_size=conv_kernels[-1],
            activation=activation,
            padding='same'
        ))
model.add(Flatten())

for _ in range(num_fc_layers):
    if activation == 'lrelu':
        model.add(Dense(
            units=fc_layer_size,
            activation=LeakyReLU(alpha=LRELU_APLHA)
        ))
    else:
        model.add(Dense(
            units=fc_layer_size,
            activation=activation
        ))

model.add(Dense(units=num_class, activation='softmax'))

# model.summary()

model.compile(loss=CategoricalCrossentropy(),
              metrics=['accuracy', Precision(), Recall()])

callback = [
    EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True
    )
]

```

```
history = model.fit(x=train_dataset,
                      epochs=num_epochs,
                      verbose=0,
                      callbacks=callback if num_epochs == 100 else None,
                      validation_data=val_dataset
                     )

val_loss, val_accuracy, val_precision, val_recall = \
    model.evaluate(val_dataset, verbose=0)

_plot_history(
    history=history,
    conv_filters=conv_filters,
    conv_kernels=conv_kernels,
    activation=activation,
    pool=pool,
    num_fc_layers=num_fc_layers
)

# convert categorical to numeric for confusion matrix plotting
y_val = [np.argmax(res) for res in np.concatenate(
    [y for x, y in val_dataset], axis=0)]
y_predict = [np.argmax(res) for res in model.predict(val_dataset)]
_plot_confusion_matrix(y_val, y_predict)

if give_model:
    return val_loss, val_accuracy, val_precision, val_recall, model
```

Task 4.1

Analyze the performance of convolution window kernel size

| Convolution Layer | Convolution kernel_size | Convolution filters size | Pooling Layers | Activation | FC layer (after Flatten) | Regularization ||-----||-----||-----|
 -----|-----|-----| 3 | (3*3, 3*3, 3*3) | [16,32,64] | Max_Pooling | Relu | 1 | Dropout of 0.1 after each layer || 3 | (3*3, 3*3, 5*5) | [16,32,64] | Max_Pooling | Relu | 1 | Dropout of
 0.1 after each layer | 3 | (3*3, 5*5, 5*5) | [16,32,64] | Max_Pooling | Relu | 1 | Dropout of 0.1 after each layer | 3 | (5*5, 5*5, 5*5) | [16,32,64] | Max_Pooling | Relu | 1 | Dropout of 0.1 after each layer |

In []: **import** pandas as pd

```
result = pd.DataFrame(columns=[  
    'Convolution kernel_size',  
    'Convolution filters size',  
    'Pooling Layers',  
    'Activation',  
    'FC layer (after Flatten)',  
    'Dropout Rate',  
    'Validation Loss',  
    'Validation Accuracy',  
    'Validation Precision',  
    'Validation Recall'  
])  
  
convolution_kernels = [  
    [(3, 3), (3, 3), (3, 3)],  
    [(3, 3), (3, 3), (5, 5)],  
    [(3, 3), (5, 5), (5, 5)],  
    [(5, 5), (5, 5), (5, 5)]]
```

```

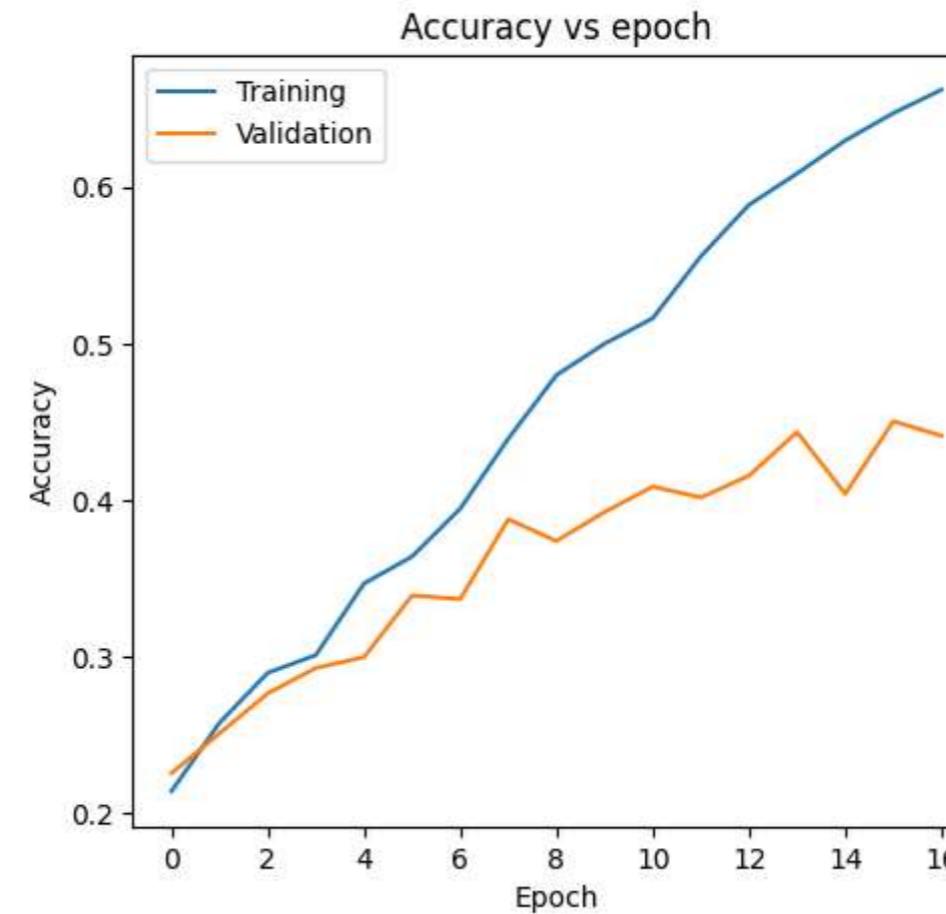
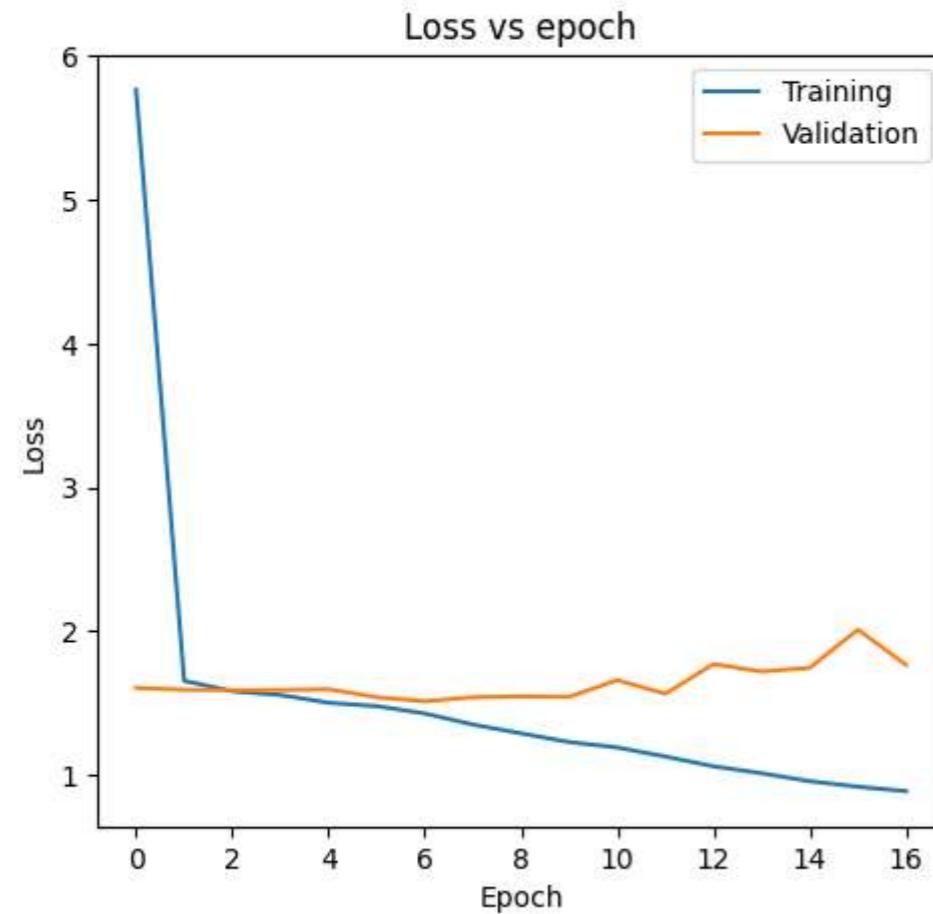
]
convolution_filters = [16, 32, 64]
activation = 'relu'
dropout_rate = 0.1
num_fc_layers = 0
pool='max'

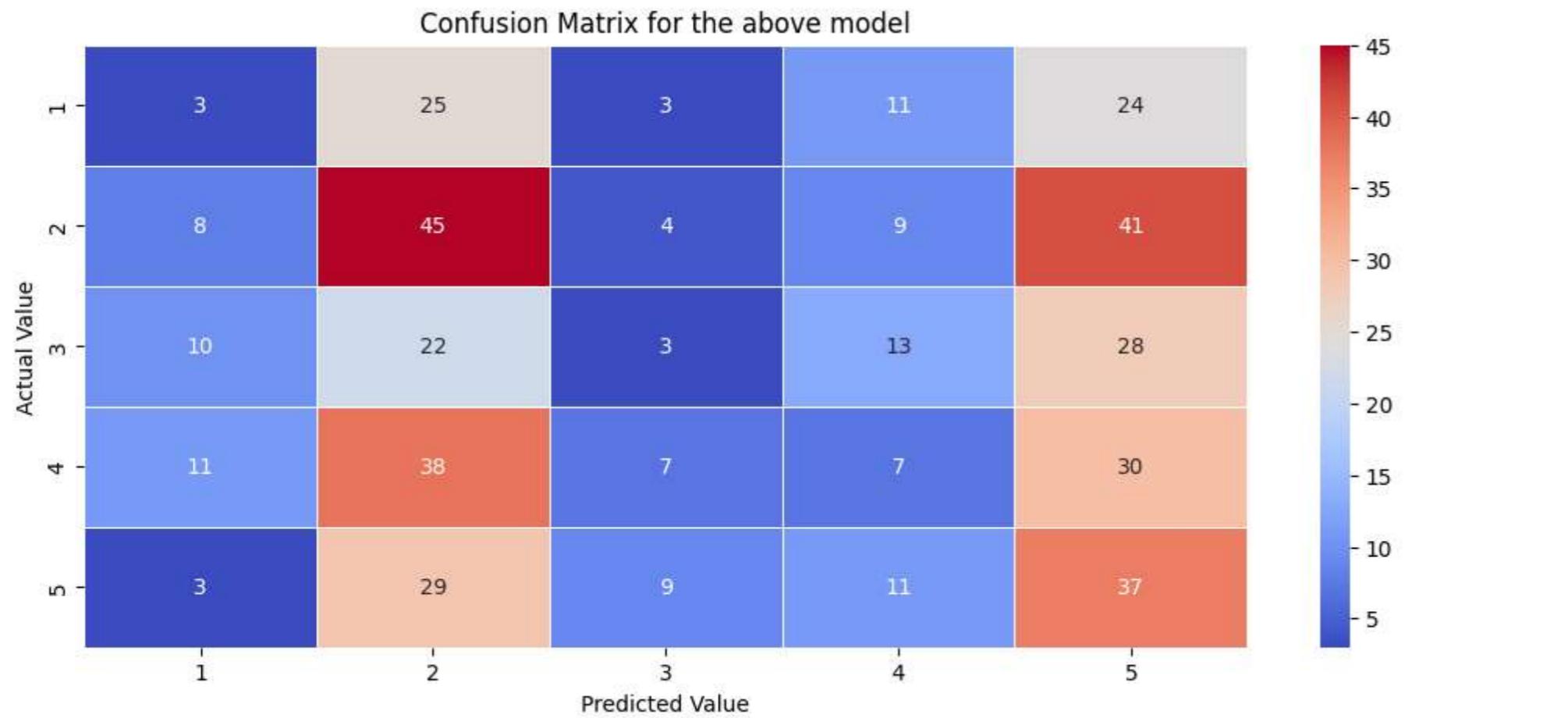
for kernels in convolution_kernels:
    val_loss, val_acc, val_precision, val_recall = train_model(
        drop_rate=dropout_rate,
        conv_kernels=kernels,
        conv_filters=convolution_filters,
        activation=activation,
        pool=pool,
        num_fc_layers=num_fc_layers,
        train_dataset=train_dataset,
        val_dataset=val_dataset
    )

    result.loc[len(result.index)] = [
        kernels,
        convolution_filters,
        pool,
        activation,
        num_fc_layers,
        dropout_rate,
        val_loss,
        val_acc,
        val_precision,
        val_recall
    ]

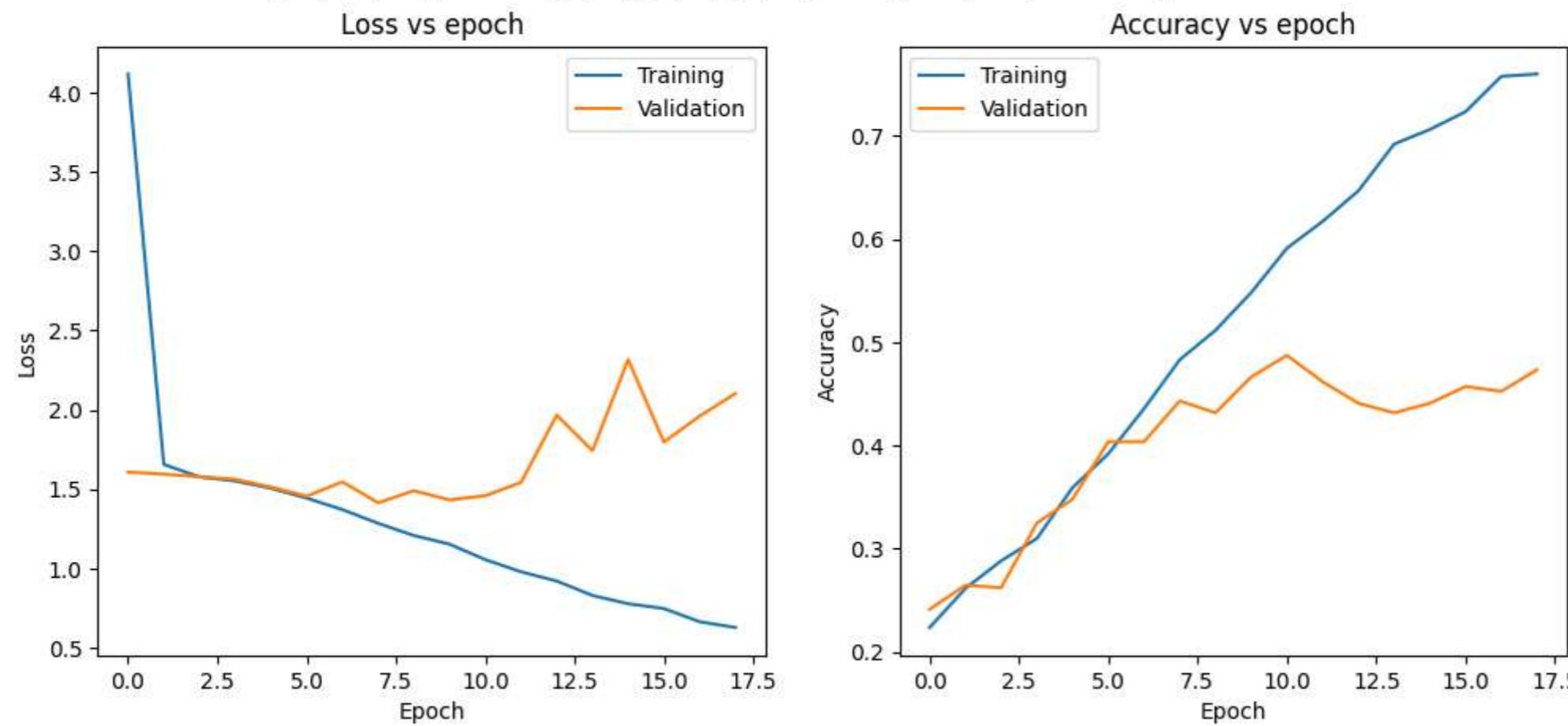
```

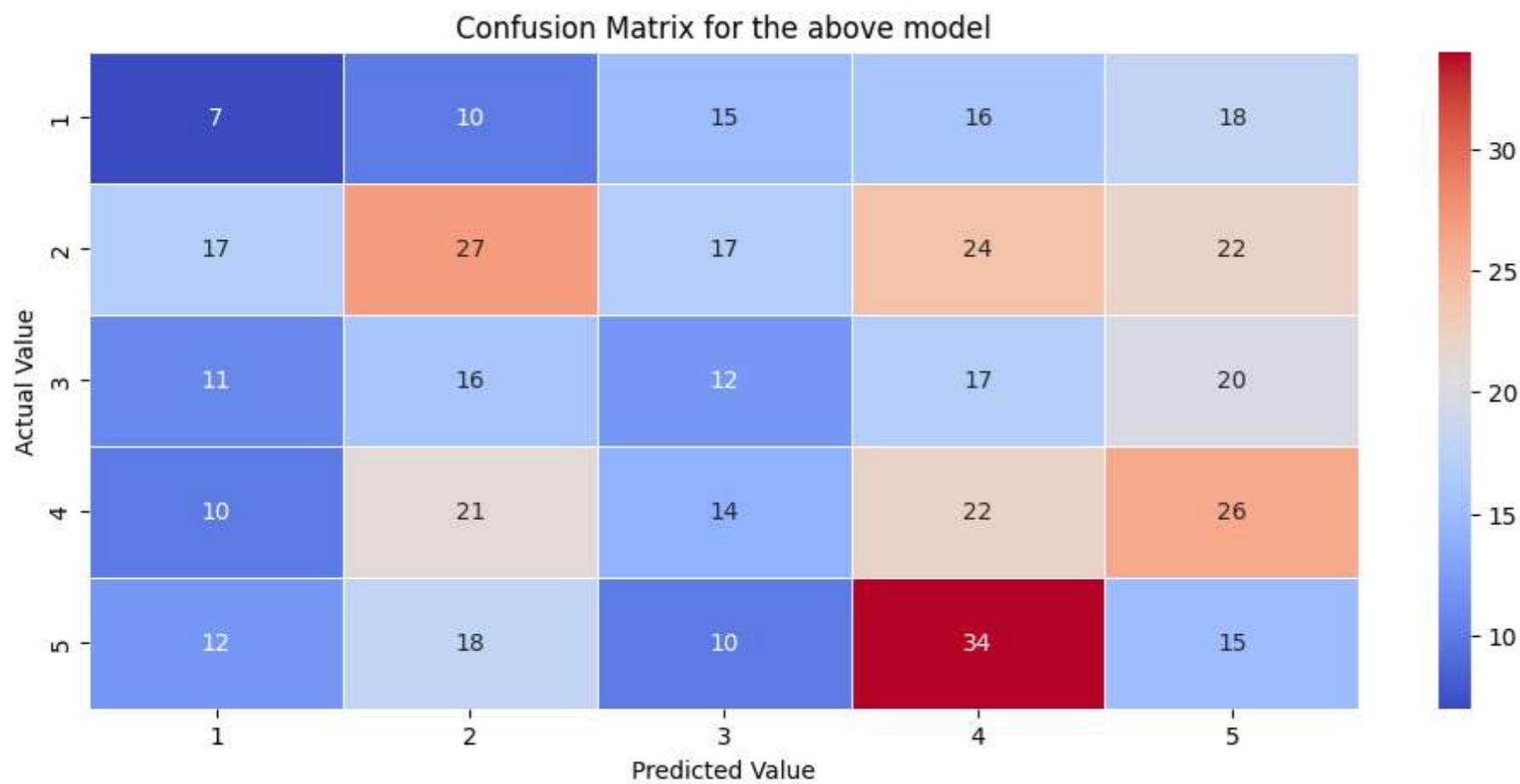
filters [16, 32, 64]; kernels [(3, 3), (3, 3), (3, 3)]; max_pool; relu; dense layers after flatten= 0



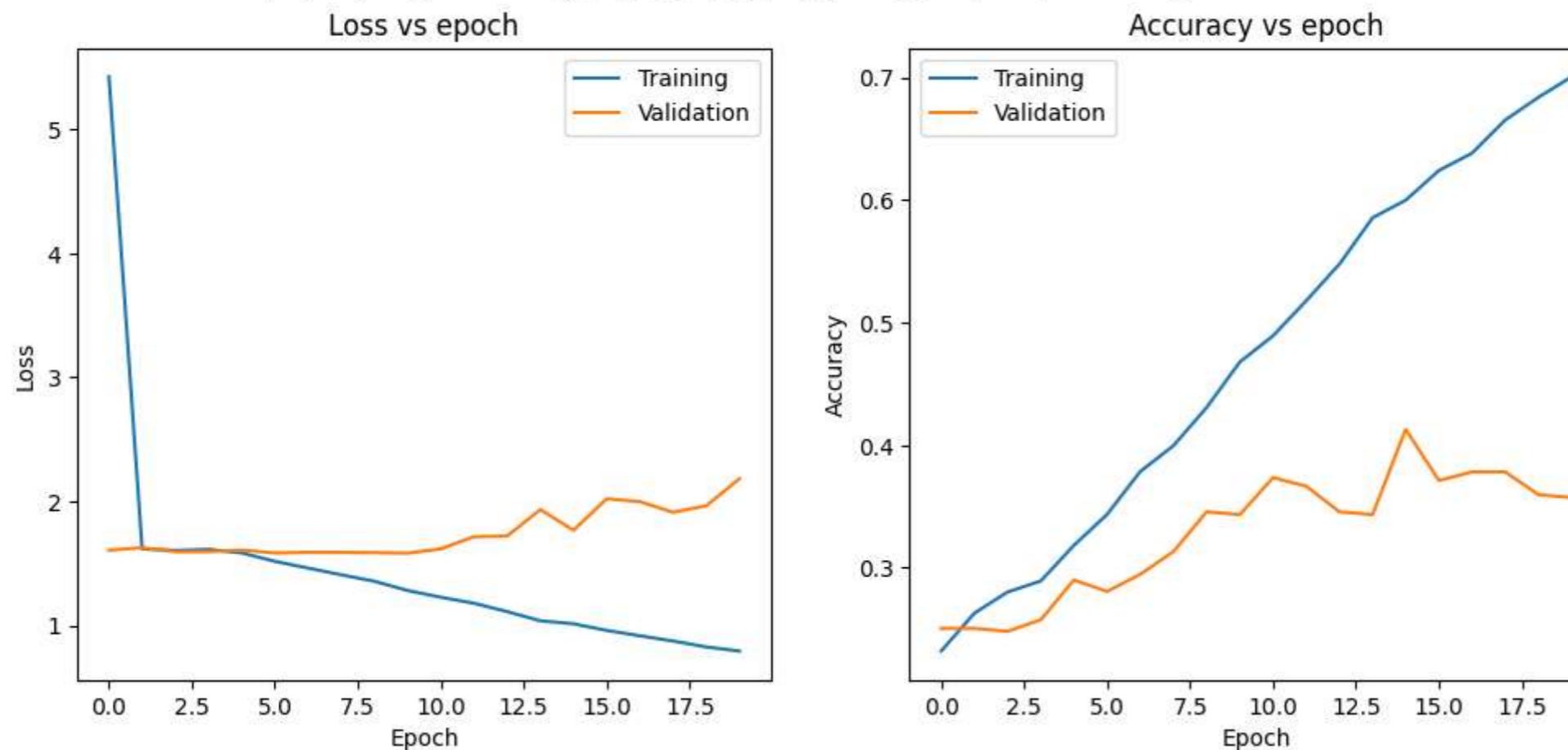


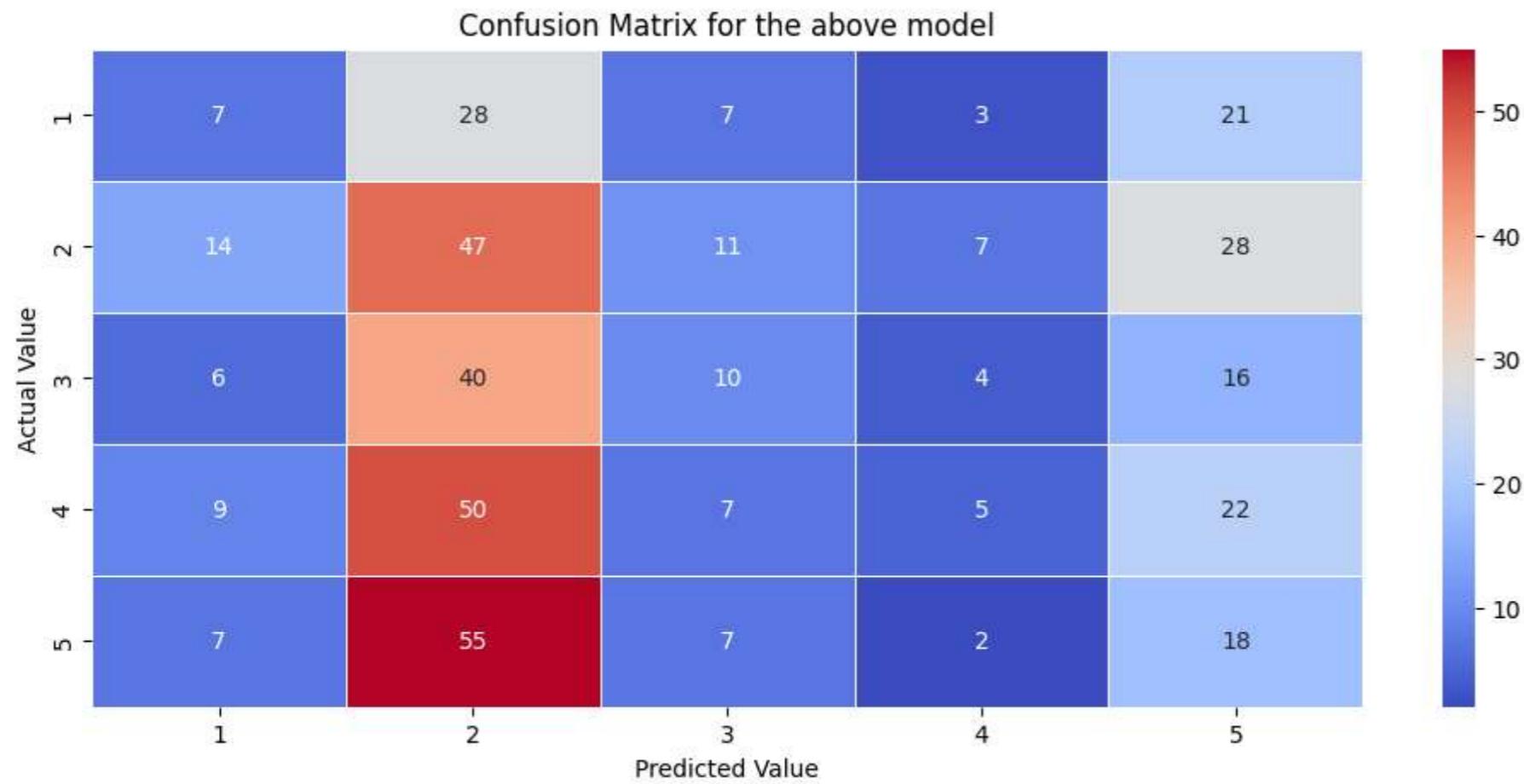
filters [16, 32, 64]; kernels [(3, 3), (3, 3), (5, 5)]; max_pool; relu; dense layers after flatten= 0



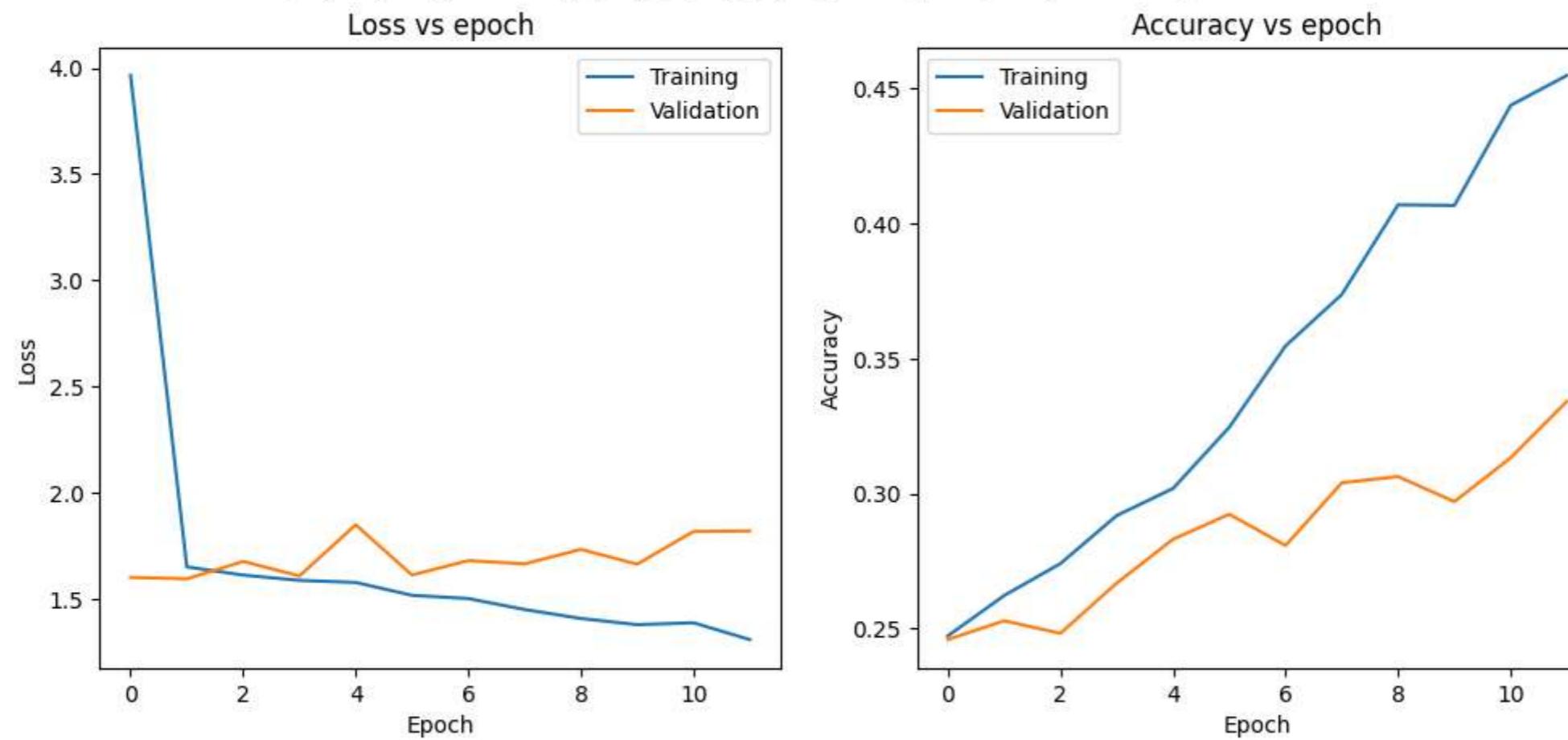


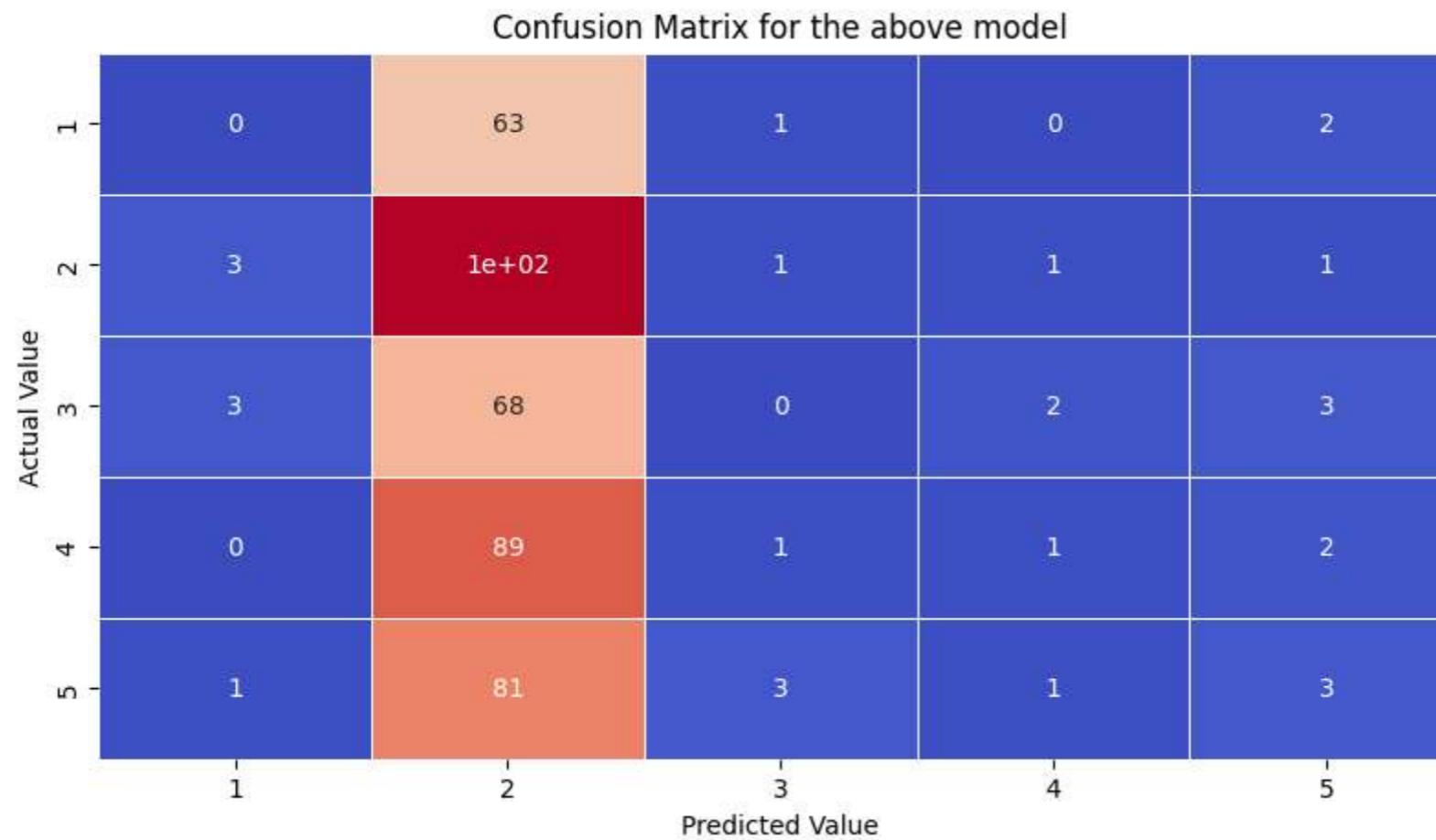
filters [16, 32, 64]; kernels [(3, 3), (5, 5), (5, 5)]; max_pool; relu; dense layers after flatten= 0





filters [16, 32, 64]; kernels [(5, 5), (5, 5), (5, 5)]; max_pool; relu; dense layers after flatten= 0





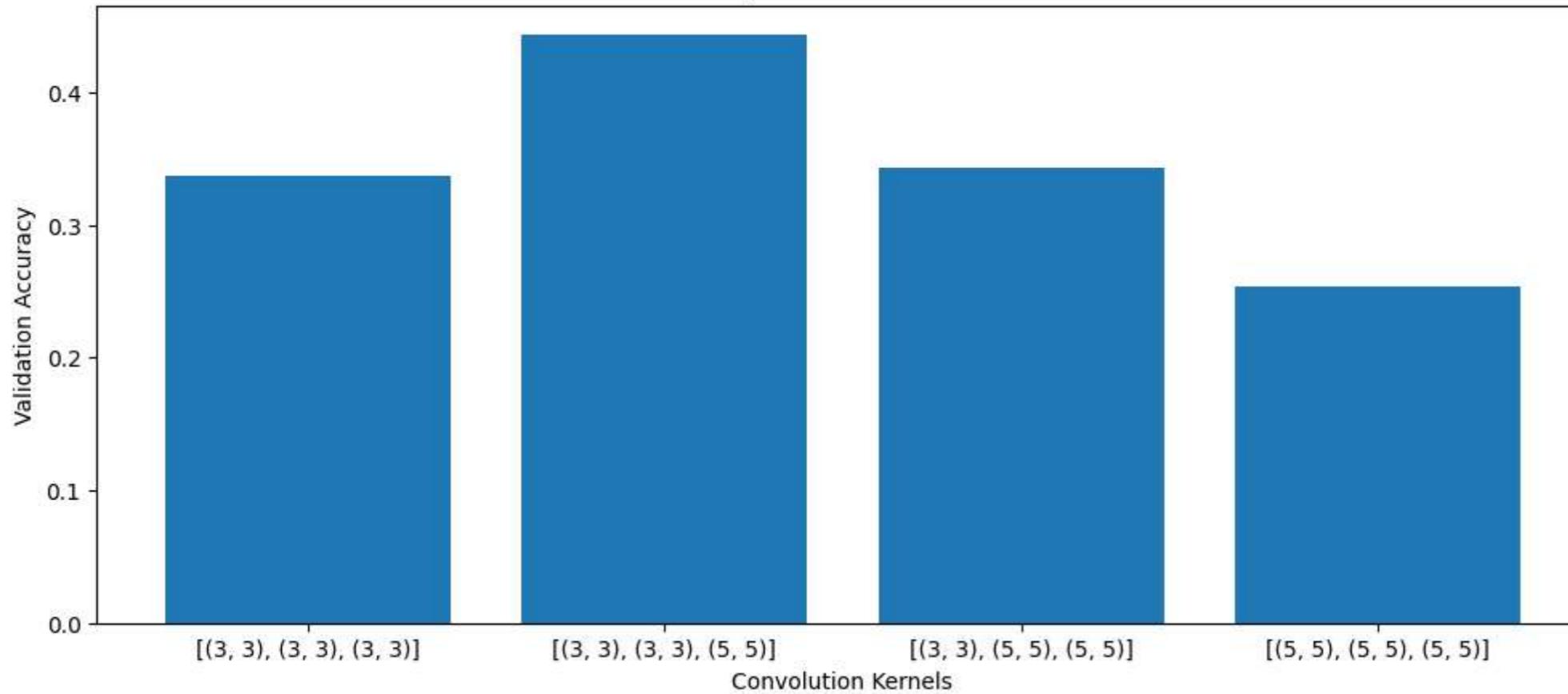
```
In [ ]: result['Validation f1'] = \
    2*result['Validation Precision']*result['Validation Recall']/(result['Validation Precision'] + result['Validation Recall'])
result
```

Out[]:

	Convolution kernel_size	Convolution filters size	Pooling Layers	Activation	FC layer (after Flatten)	Dropout Rate	Validation Loss	Validation Accuracy	Validation Precision	Validation Recall	Validation f1
0	[(3, 3), (3, 3), (3, 3)]	[16, 32, 64]	max	relu	0	0.1	1.513770	0.336427	0.617021	0.067285	0.121339
1	[(3, 3), (3, 3), (5, 5)]	[16, 32, 64]	max	relu	0	0.1	1.413185	0.443155	0.653846	0.157773	0.254206
2	[(3, 3), (5, 5), (5, 5)]	[16, 32, 64]	max	relu	0	0.1	1.583565	0.343387	0.528090	0.109049	0.180769
3	[(5, 5), (5, 5), (5, 5)]	[16, 32, 64]	max	relu	0	0.1	1.594527	0.252900	0.500000	0.002320	0.004619

```
In [ ]: plt.figure(figsize=(12, 5))
plt.bar(
    [str(res) for res in result['Convolution kernel_size']],
    result['Validation Accuracy'])
plt.ylabel('Validation Accuracy')
plt.xlabel('Convolution Kernels')
plt.title("Validation Accuracy for different Convolution Kernel")
plt.show()
```

Validation Accuracy for different Convolution Kernel



```
In [ ]: best_conv_kernel = result.sort_values(  
      by=['Validation Accuracy', 'Validation Loss'],  
      ascending=[False, True]  
)['Convolution kernel_size'].iloc[0]  
  
best_conv_kernel
```

```
Out[ ]: [(3, 3), (3, 3), (5, 5)]
```

Task 4.2

For the best set of parameters obtained above, use two and three FC layers(After Flatten).

```
In [ ]: result = pd.DataFrame(columns=[  
    'Convolution kernel_size',  
    'Convolution filters size',  
    'Pooling Layers',  
    'Activation',  
    'FC layer (after Flatten)',  
    'Dropout Rate',  
    'Validation Loss',  
    'Validation Accuracy',  
    'Validation Precision',  
    'Validation Recall'  
])  
  
convolution_filters = [16, 32, 64]  
activation = 'relu'  
dropout_rate = 0.1  
num_fc_layers = [1, 2, 3]  
pool = 'max'
```

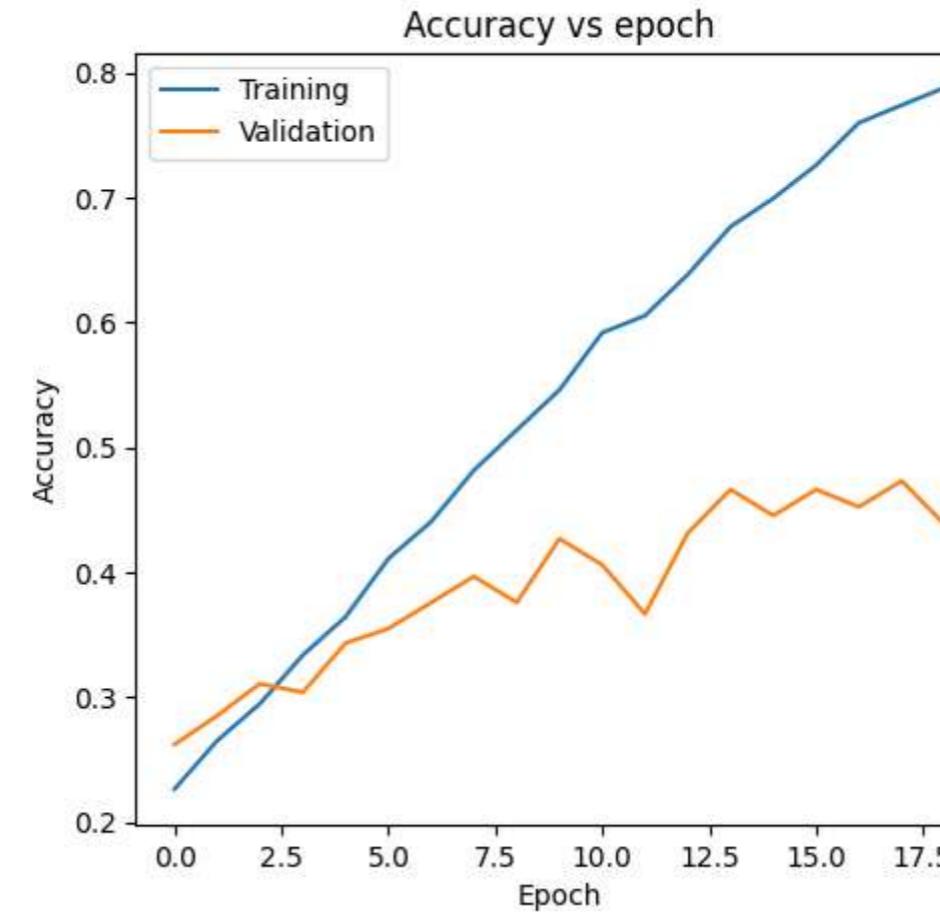
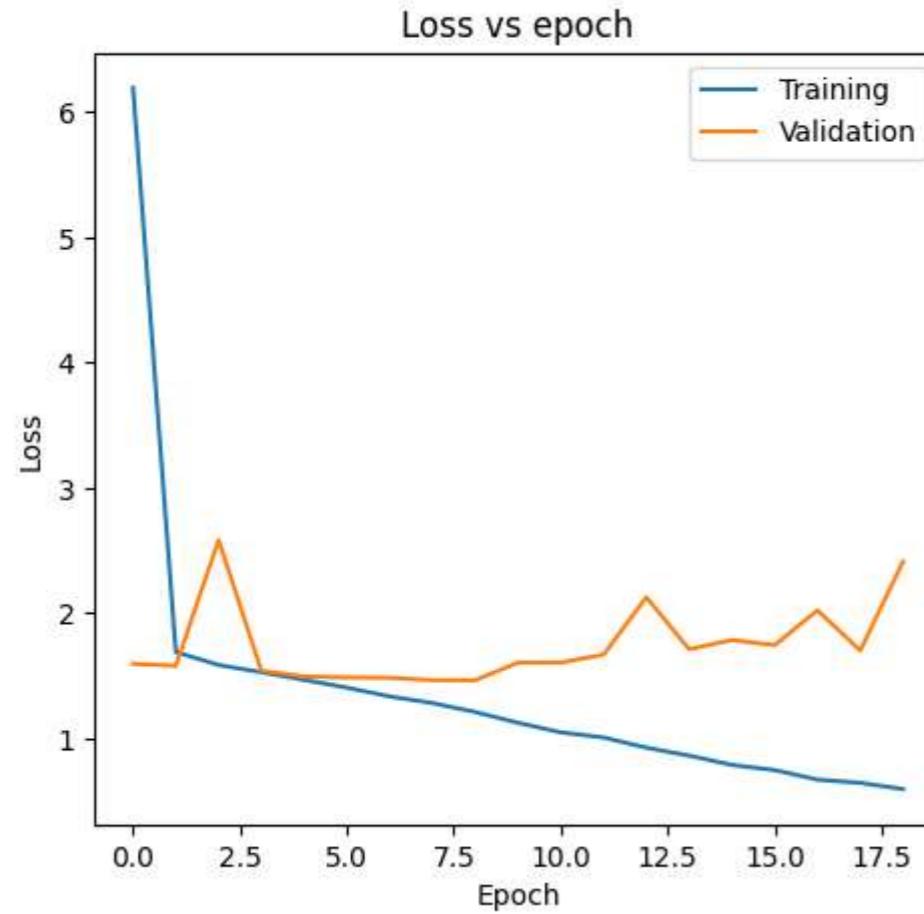
```

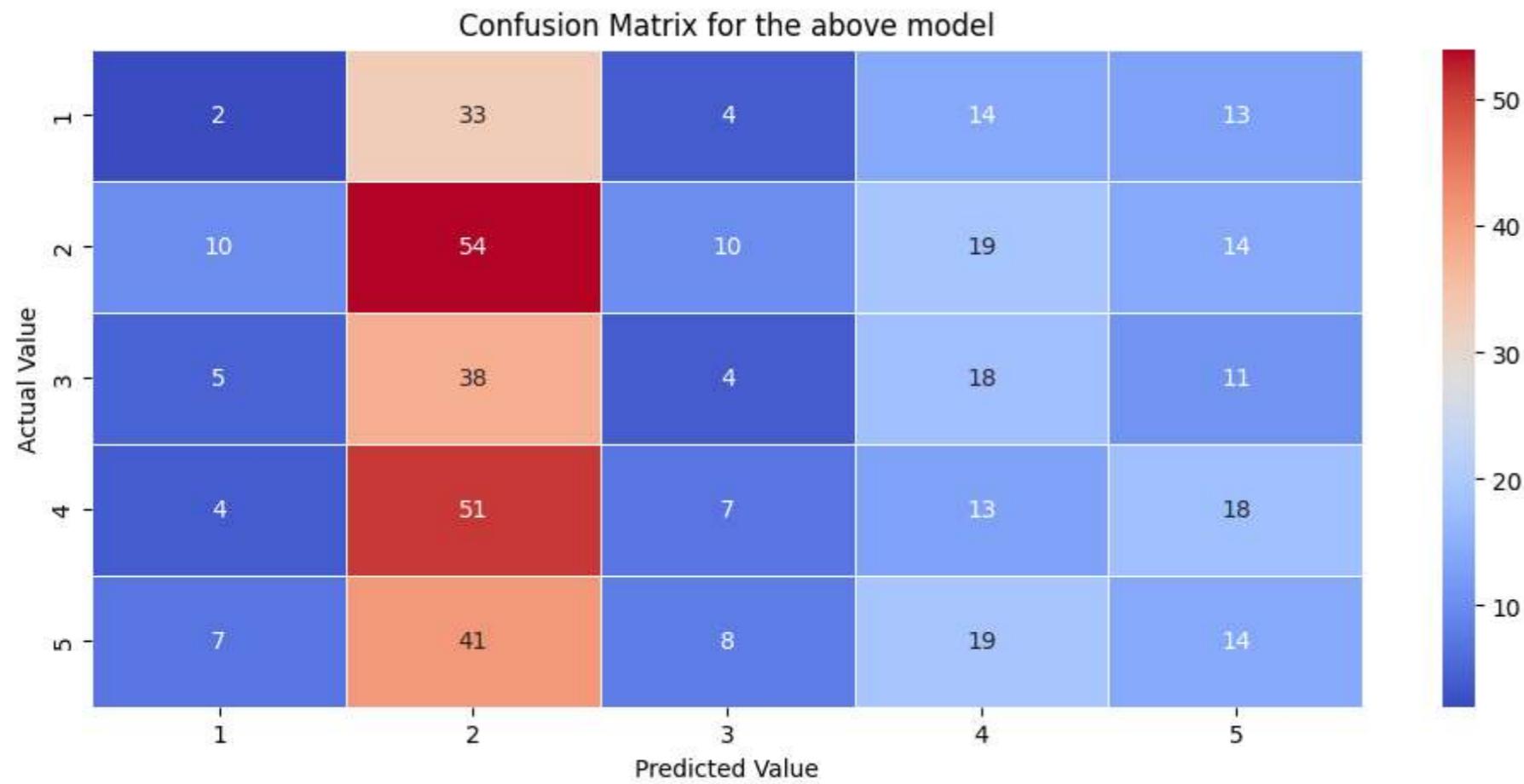
for layers in num_fc_layers:
    val_loss, val_acc, val_precision, val_recall = train_model(
        drop_rate=dropout_rate,
        conv_kernels=best_conv_kernel,
        conv_filters=convolution_filters,
        activation=activation,
        pool=pool,
        num_fc_layers=layers,
        train_dataset=train_dataset,
        val_dataset=val_dataset
    )

    result.loc[len(result.index)] = [
        best_conv_kernel,
        convolution_filters,
        pool,
        activation,
        layers,
        dropout_rate,
        val_loss,
        val_acc,
        val_precision,
        val_recall
    ]
]

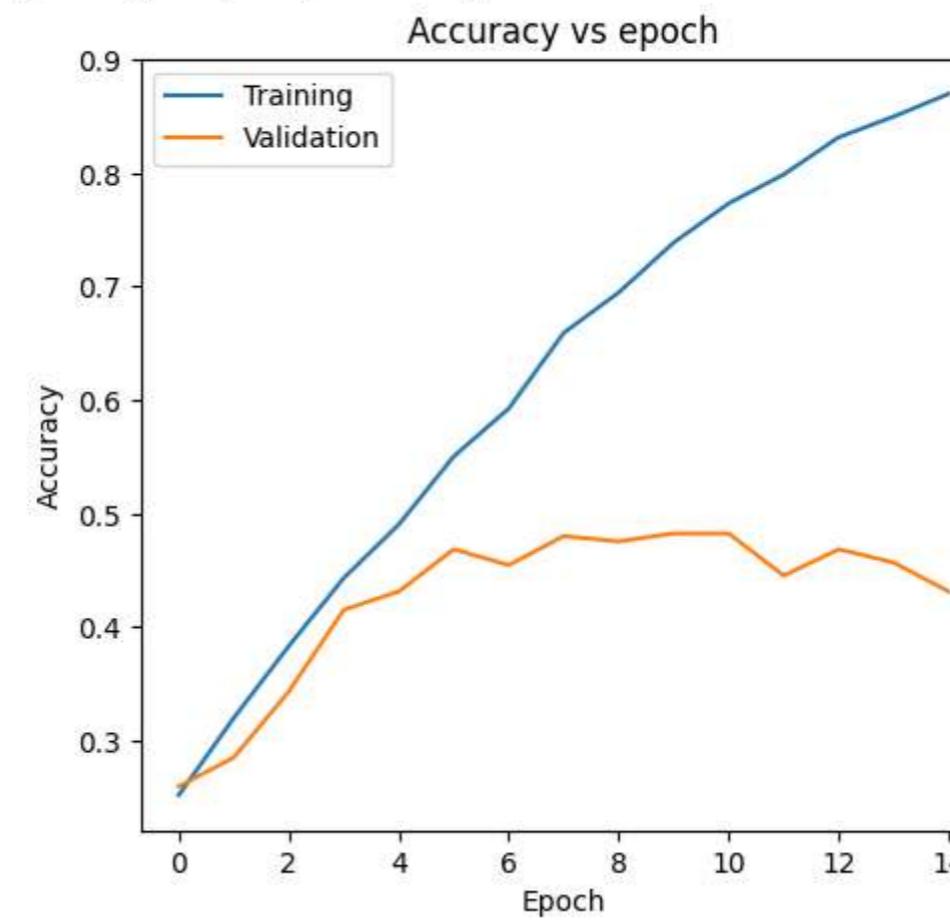
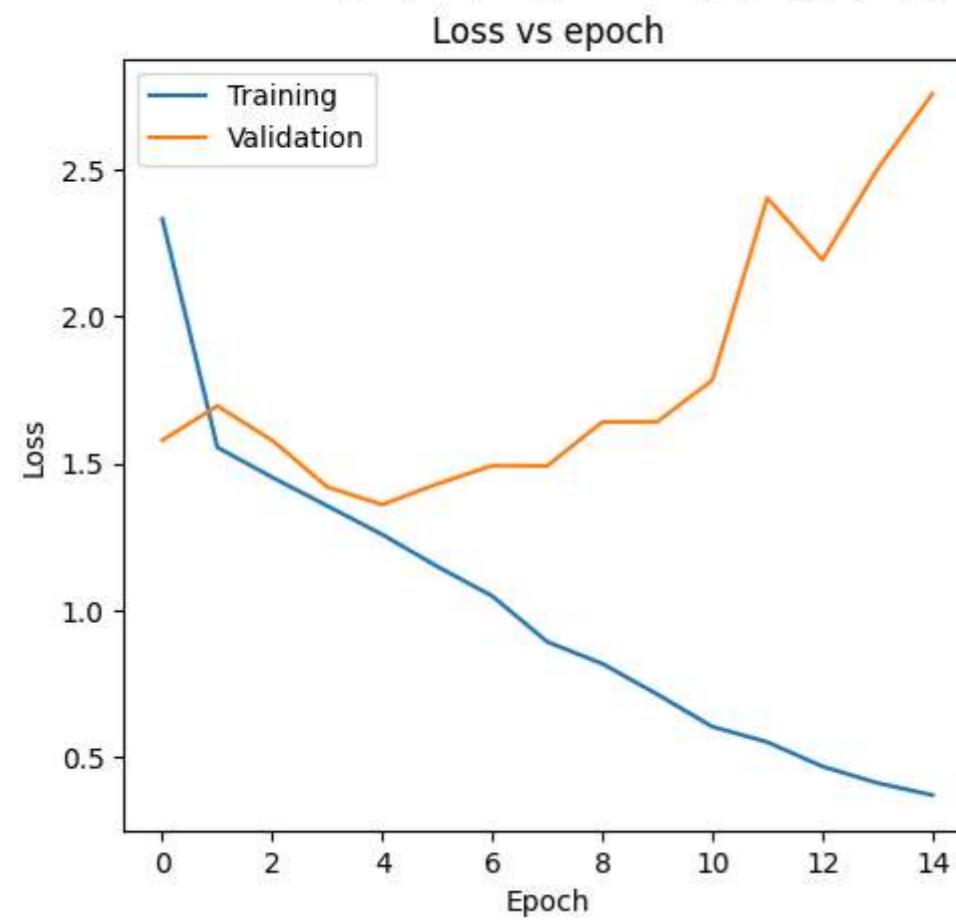
```

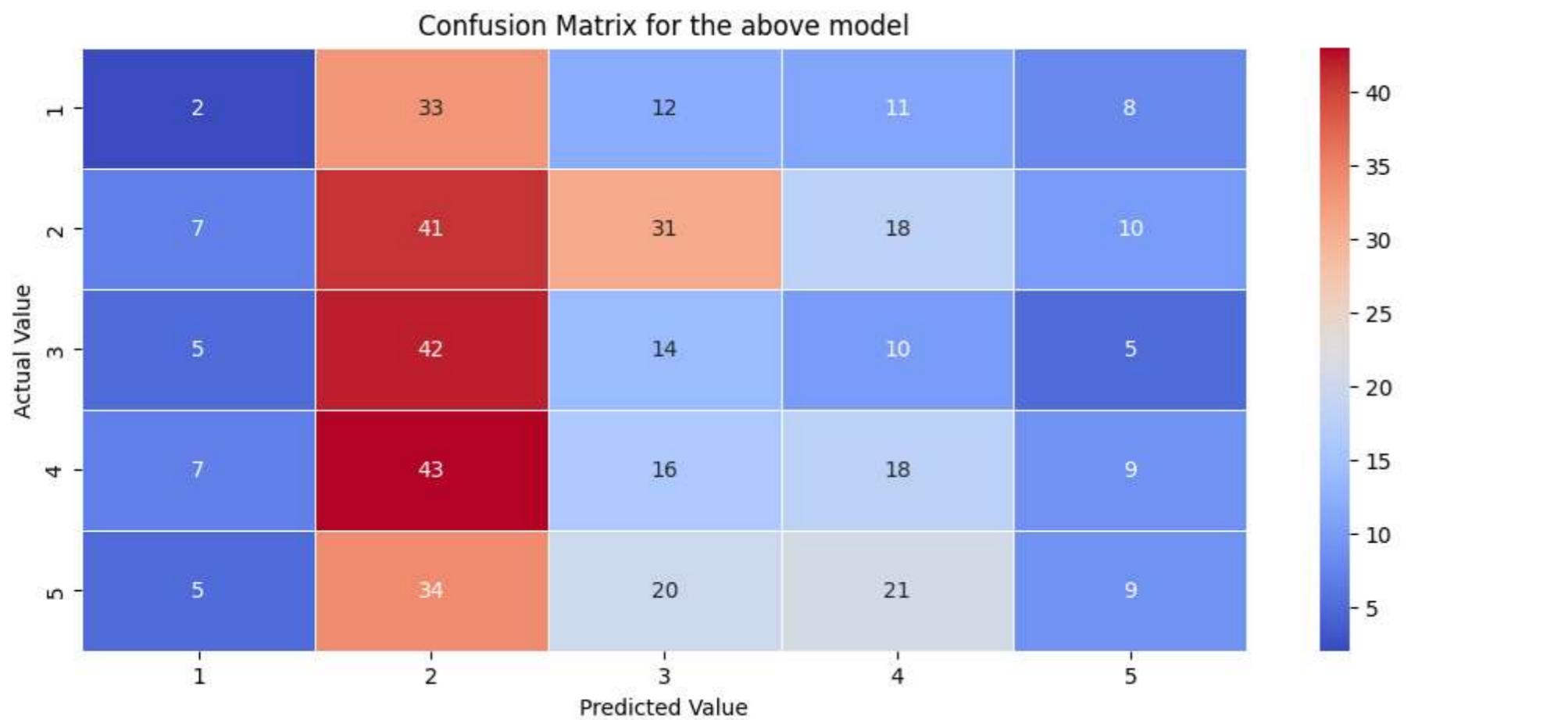
filters [16, 32, 64]; kernels [(3, 3), (3, 3), (5, 5)]; max_pool; relu; dense layers after flatten= 1



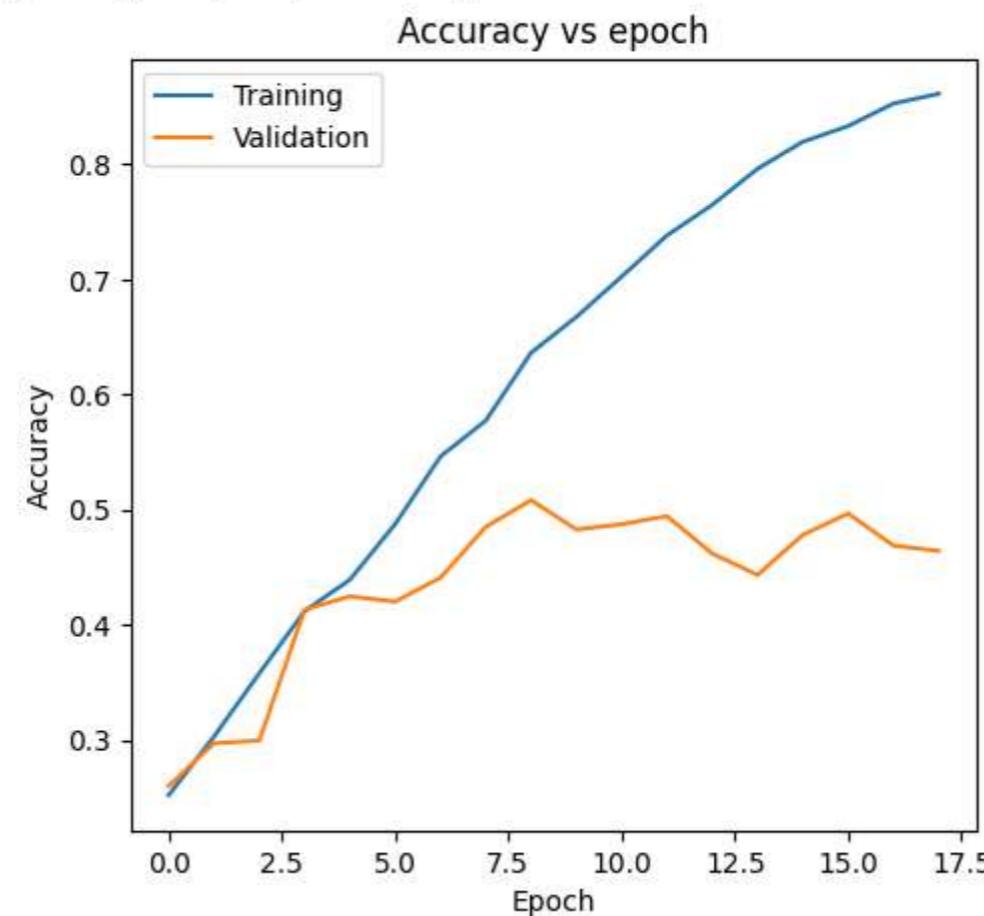
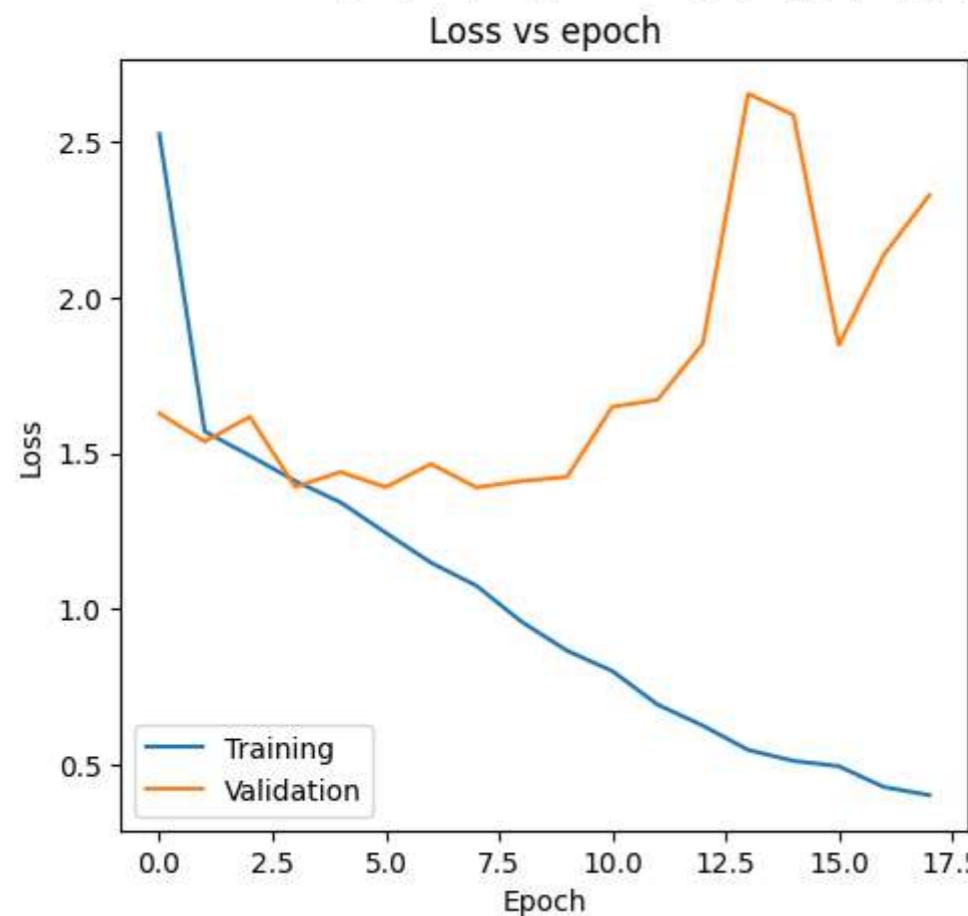


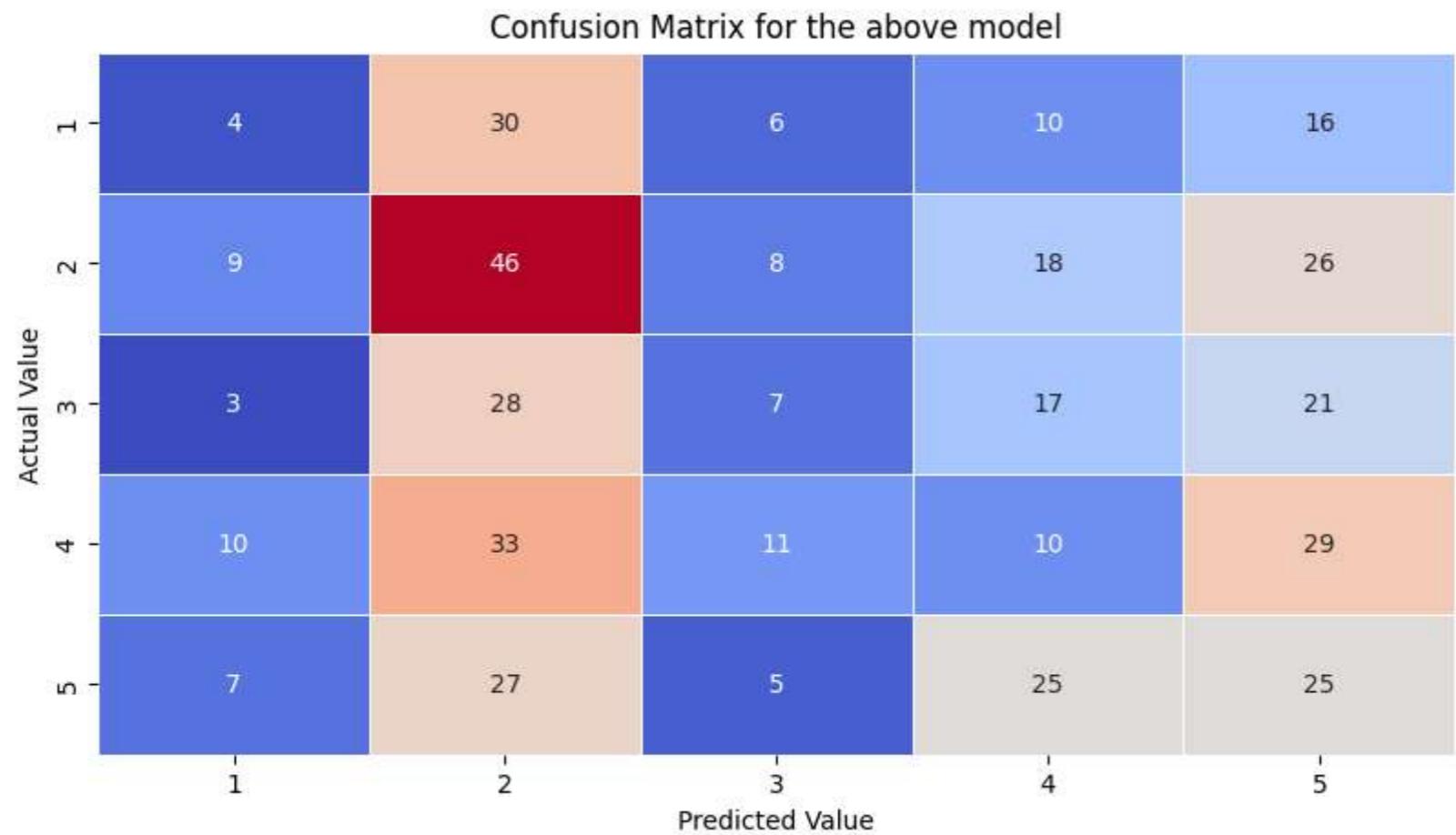
filters [16, 32, 64]; kernels [(3, 3), (3, 3), (5, 5)]; max_pool; relu; dense layers after flatten= 2





filters [16, 32, 64]; kernels [(3, 3), (3, 3), (5, 5)]; max_pool; relu; dense layers after flatten= 3



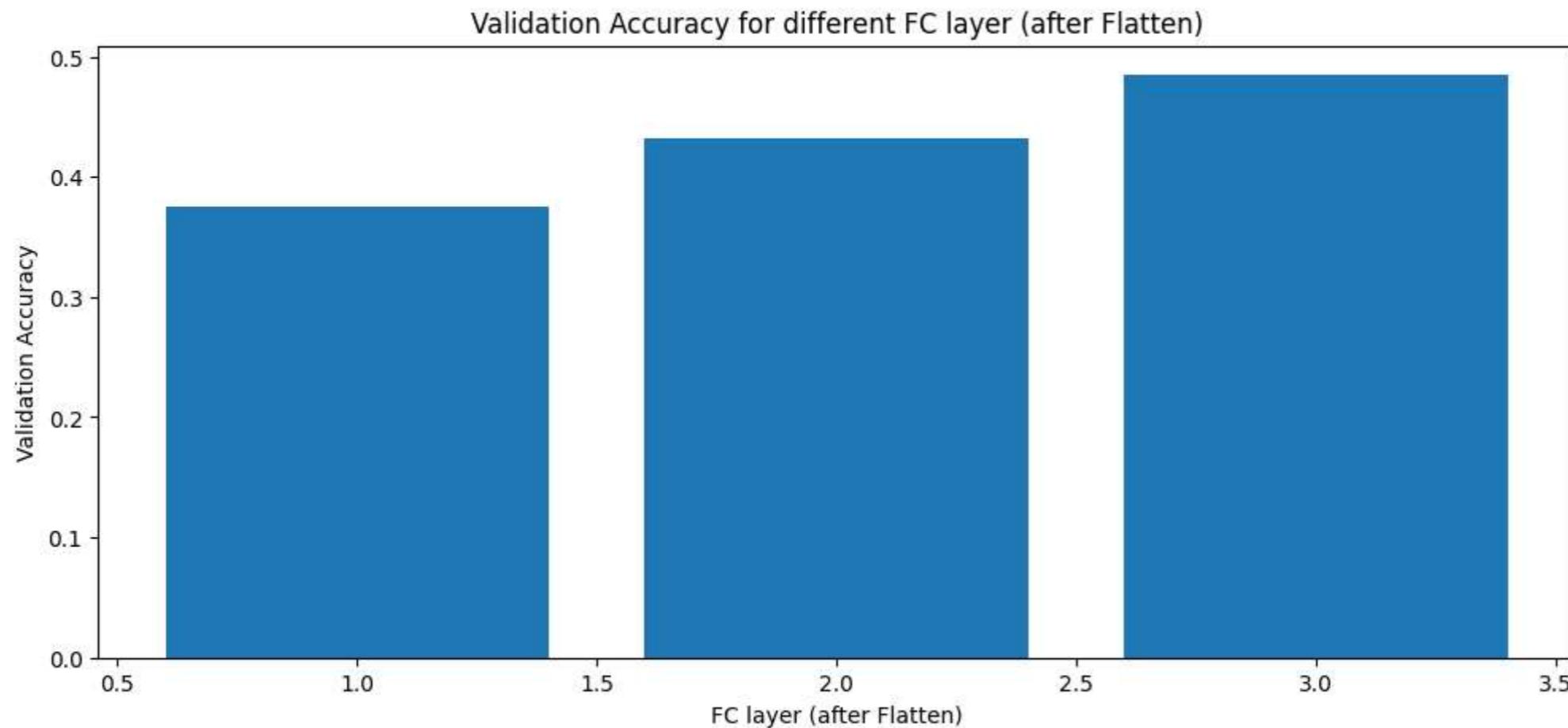


```
In [ ]: result['Validation f1'] = \
    2*result['Validation Precision']*result['Validation Recall']/(result['Validation Precision'] + result['Validation Recall'])
result
```

Out[]:

	Convolution kernel_size	Convolution filters size	Pooling Layers	Activation	FC layer (after Flatten)	Dropout Rate	Validation Loss	Validation Accuracy	Validation Precision	Validation Recall	Validation f1
0	[(3, 3), (3, 3), (5, 5)]	[16, 32, 64]	max	relu	1	0.1	1.465557	0.375870	0.543478	0.174014	0.263620
1	[(3, 3), (3, 3), (5, 5)]	[16, 32, 64]	max	relu	2	0.1	1.358761	0.431555	0.625000	0.197216	0.299824
2	[(3, 3), (3, 3), (5, 5)]	[16, 32, 64]	max	relu	3	0.1	1.390345	0.484919	0.591489	0.322506	0.417417

```
In [ ]: plt.figure(figsize=(12, 5))
plt.bar(
    result['FC layer (after Flatten)'],
    result['Validation Accuracy'])
plt.ylabel('Validation Accuracy')
plt.xlabel('FC layer (after Flatten)')
plt.title("Validation Accuracy for different FC layer (after Flatten)")
plt.show()
```



```
In [ ]: best_num_fc_layers = result.sort_values(  
        by=['Validation Accuracy', 'Validation Loss'],  
        ascending=[False, True]  
       )['FC layer (after Flatten)'].iloc[0]  
  
best_num_fc_layers
```

```
Out[ ]: 3
```

Task 4.3

For the best set of parameters obtained above, use average pooling instead of Max pooling.

```
In [ ]: result = pd.DataFrame(columns=[  
        'Convolution kernel_size',  
        'Convolution filters size',  
        'Pooling Layers',  
        'Activation',  
        'FC layer (after Flatten)',  
        'Dropout Rate',  
        'Validation Loss',  
        'Validation Accuracy',  
        'Validation Precision',  
        'Validation Recall'  
    ])  
  
convolution_filters = [16, 32, 64]  
activation = 'relu'  
dropout_rate = 0.1  
pools = ['max', 'avg']
```

```

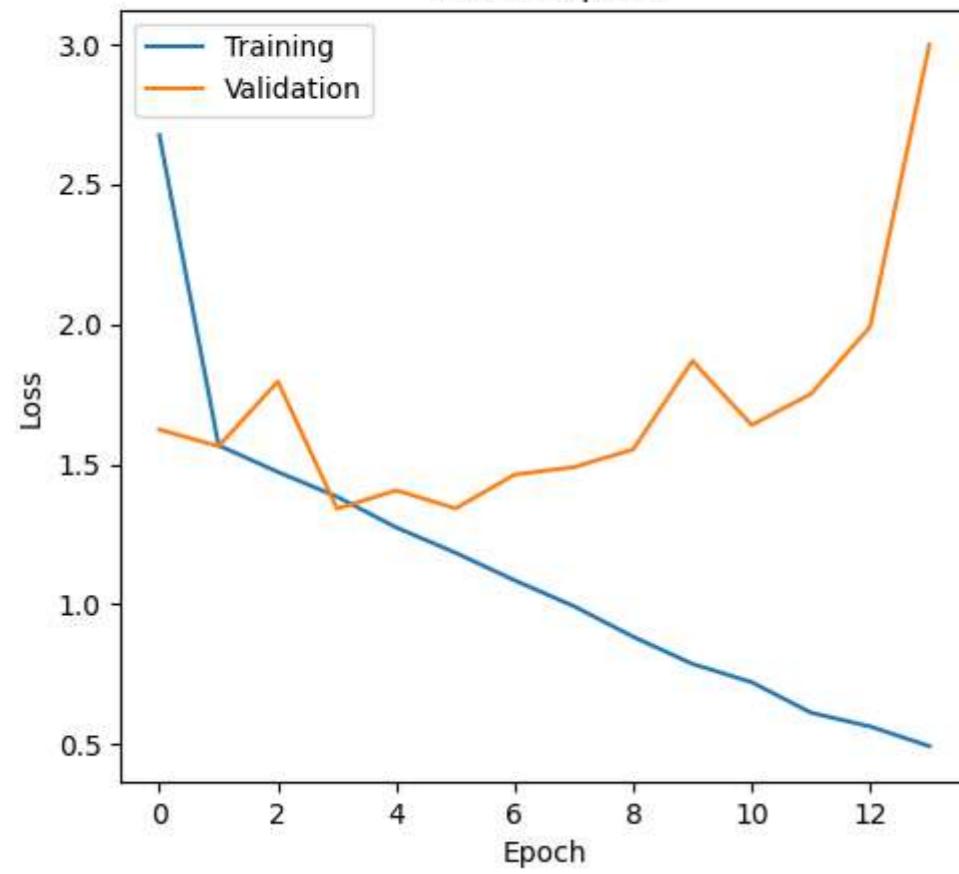
for pool in pools:
    val_loss, val_acc, val_precision, val_recall = train_model(
        drop_rate=dropout_rate,
        conv_kernels=best_conv_kernel,
        conv_filters=convolution_filters,
        activation=activation,
        pool=pool,
        num_fc_layers=best_num_fc_layers,
        train_dataset=train_dataset,
        val_dataset=val_dataset
    )

    result.loc[len(result.index)] = [
        best_conv_kernel,
        convolution_filters,
        pool,
        activation,
        best_num_fc_layers,
        dropout_rate,
        val_loss,
        val_acc,
        val_precision,
        val_recall
    ]

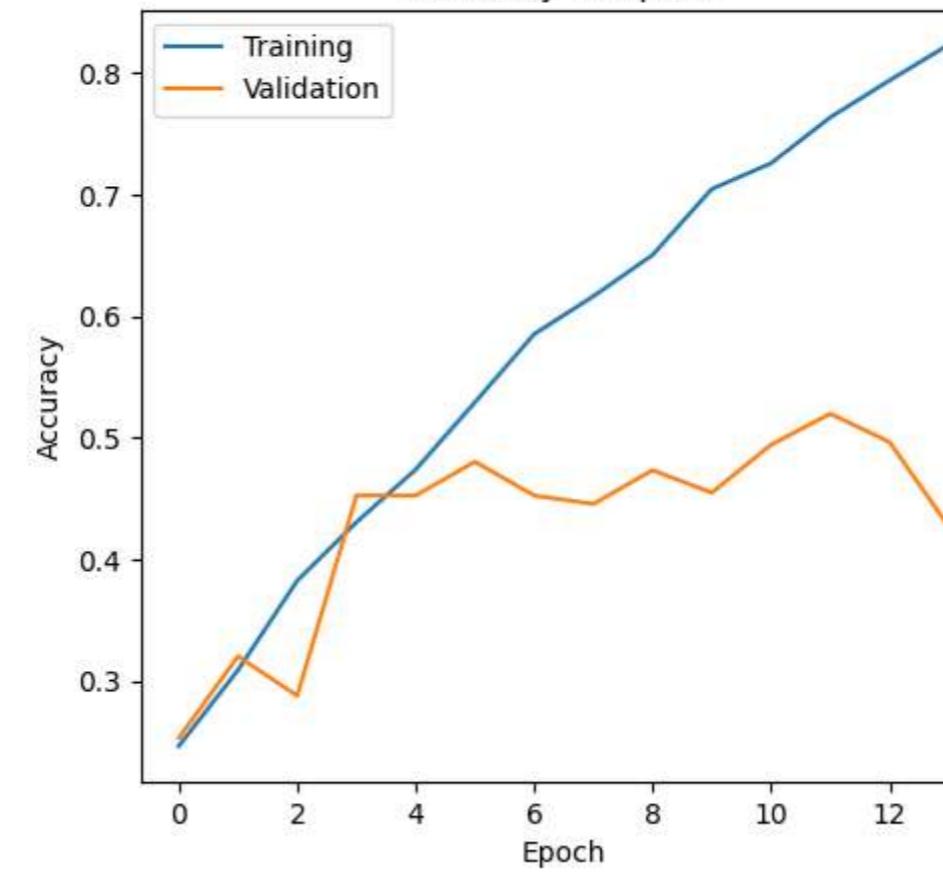
```

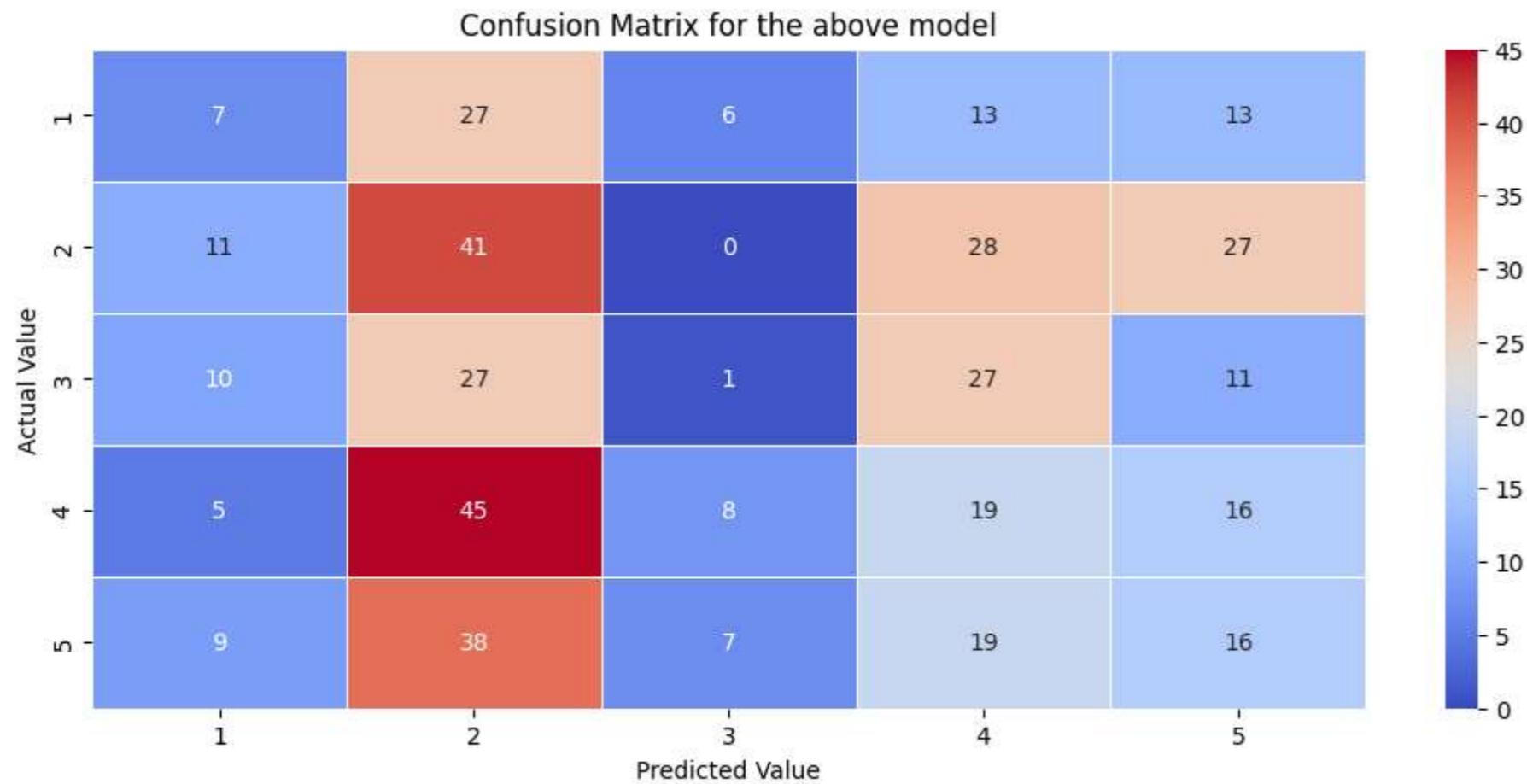
filters [16, 32, 64]; kernels [(3, 3), (3, 3), (5, 5)]; max_pool; relu; dense layers after flatten= 3

Loss vs epoch

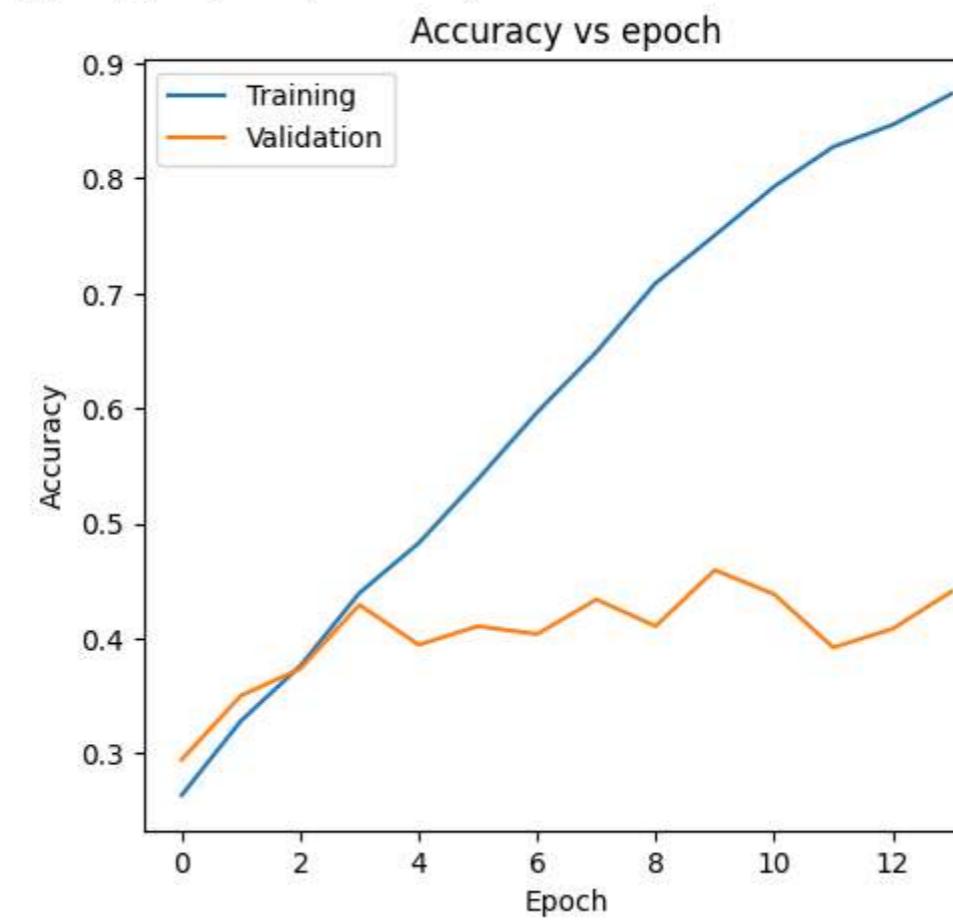
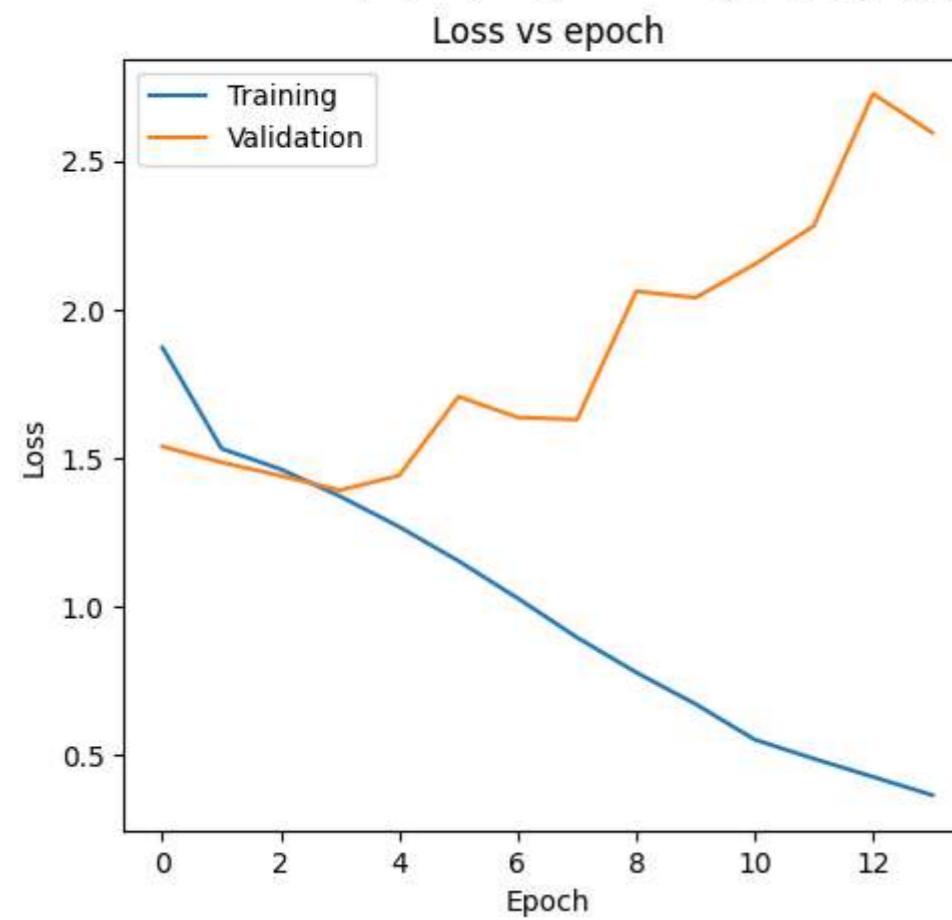


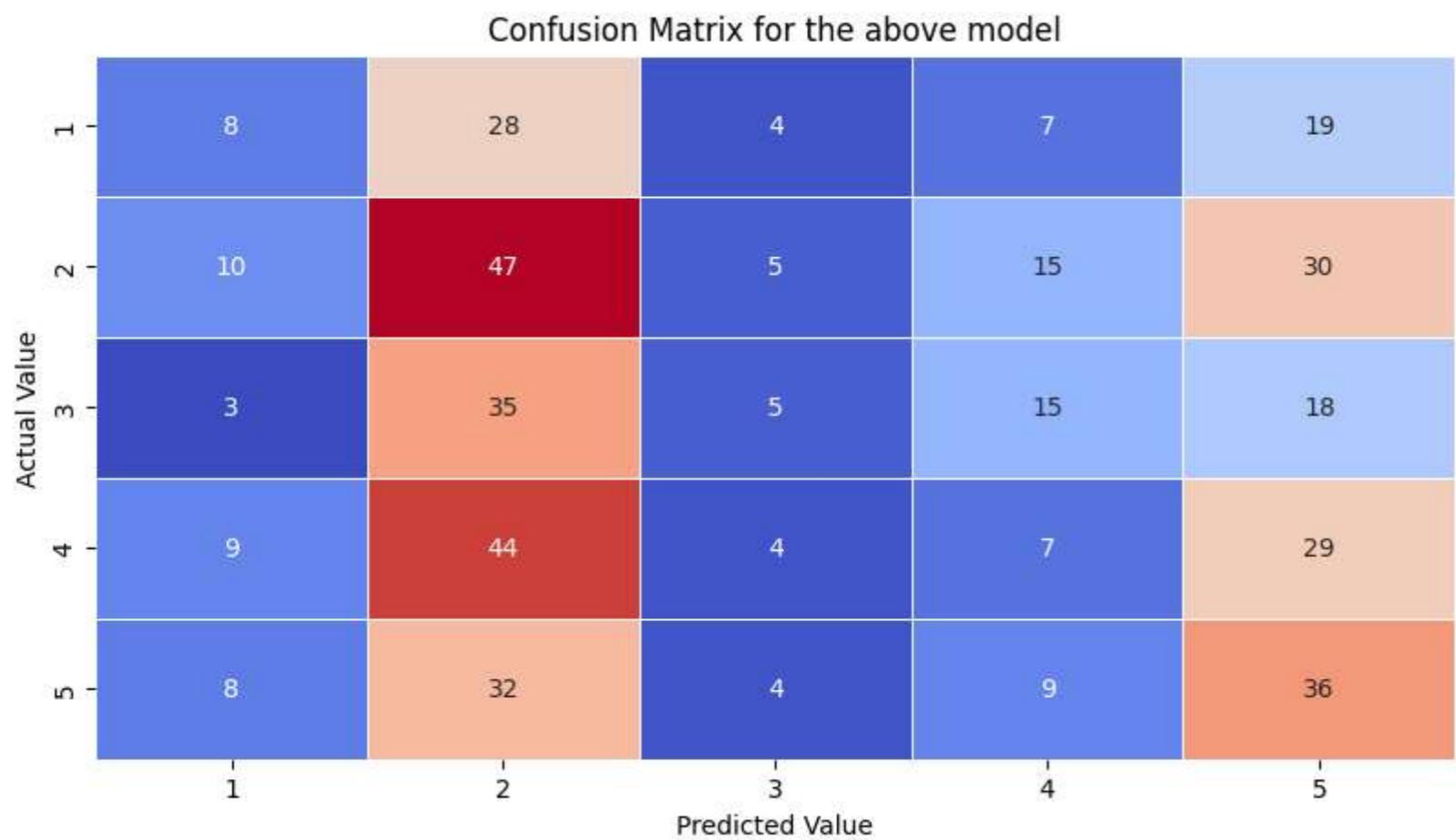
Accuracy vs epoch





filters [16, 32, 64]; kernels [(3, 3), (3, 3), (5, 5)]; avg_pool; relu; dense layers after flatten= 3



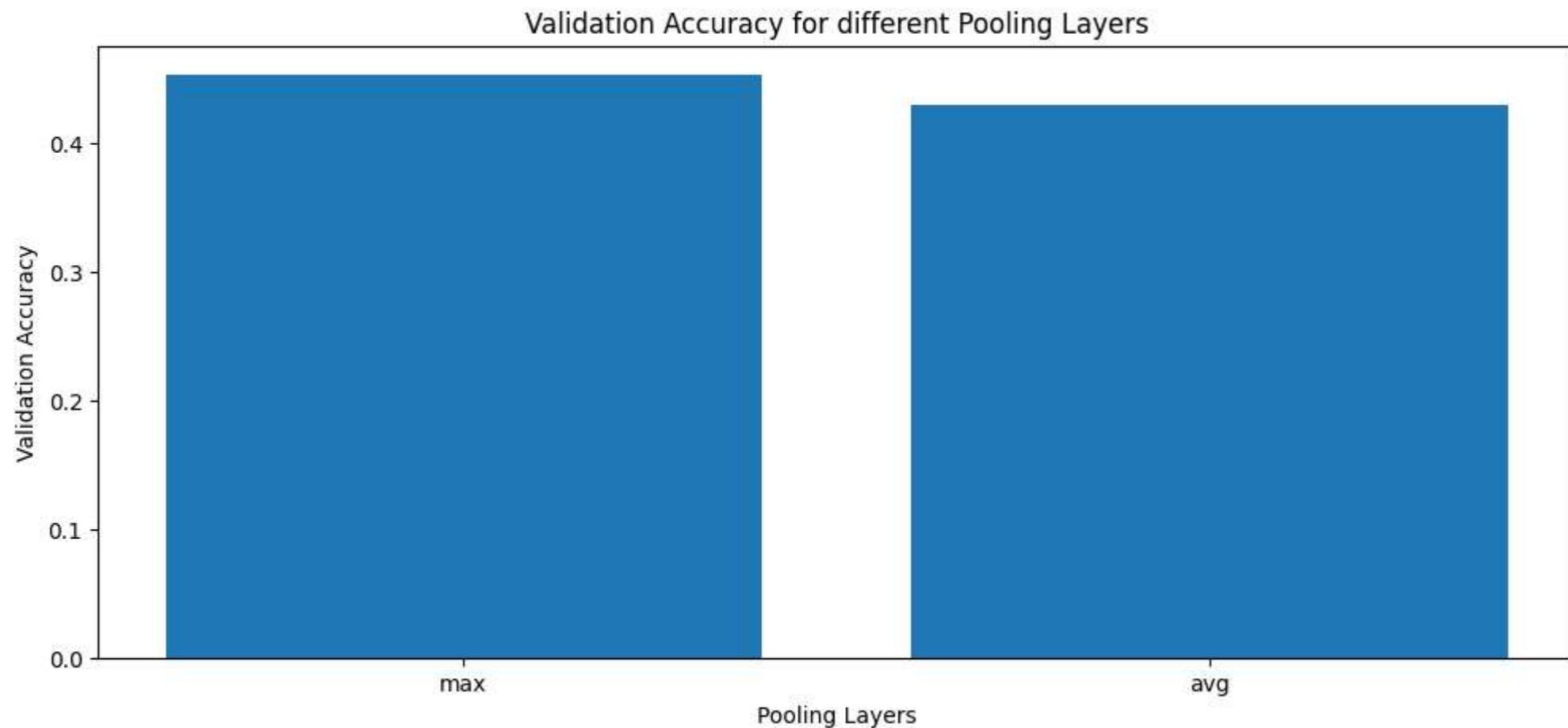


```
In [ ]: result['Validation f1'] = \
    2*result['Validation Precision']*result['Validation Recall']/(result['Validation Precision'] + result['Validation Recall'])
result
```

Out[]:

	Convolution kernel_size	Convolution filters size	Pooling Layers	Activation	FC layer (after Flatten)	Dropout Rate	Validation Loss	Validation Accuracy	Validation Precision	Validation Recall	Validation f1
0	[(3, 3), (3, 3), (5, 5)]	[16, 32, 64]	max	relu		3	0.1	1.341094	0.452436	0.752688	0.162413
1	[(3, 3), (3, 3), (5, 5)]	[16, 32, 64]	avg	relu		3	0.1	1.390802	0.429234	0.622222	0.129930

```
In [ ]: plt.figure(figsize=(12, 5))
plt.bar(
    result['Pooling Layers'],
    result['Validation Accuracy'])
plt.ylabel('Validation Accuracy')
plt.xlabel('Pooling Layers')
plt.title("Validation Accuracy for different Pooling Layers")
plt.show()
```



```
In [ ]: best_pool = result.sort_values(
    by=['Validation Accuracy', 'Validation Loss'],
    ascending=[False, True]
)[['Pooling Layers'].iloc[0]]
```

```
best_pool
```

```
Out[ ]: 'max'
```

Task 4.4

For the best set of parameters obtained above, use the activation function: Sigmoid, ELU, Leaky Relu (alpha= 0.01)

```
In [ ]: result = pd.DataFrame(columns=[
    'Convolution kernel_size',
    'Convolution filters size',
    'Pooling Layers',
    'Activation',
    'FC layer (after Flatten)',
    'Dropout Rate',
    'Validation Loss',
    'Validation Accuracy',
    'Validation Precision',
    'Validation Recall'
])

convolution_filters = [16, 32, 64]
activations = ['sigmoid', 'elu', 'relu', 'lrelu']
dropout_rate = 0.1

for activation in activations:
```

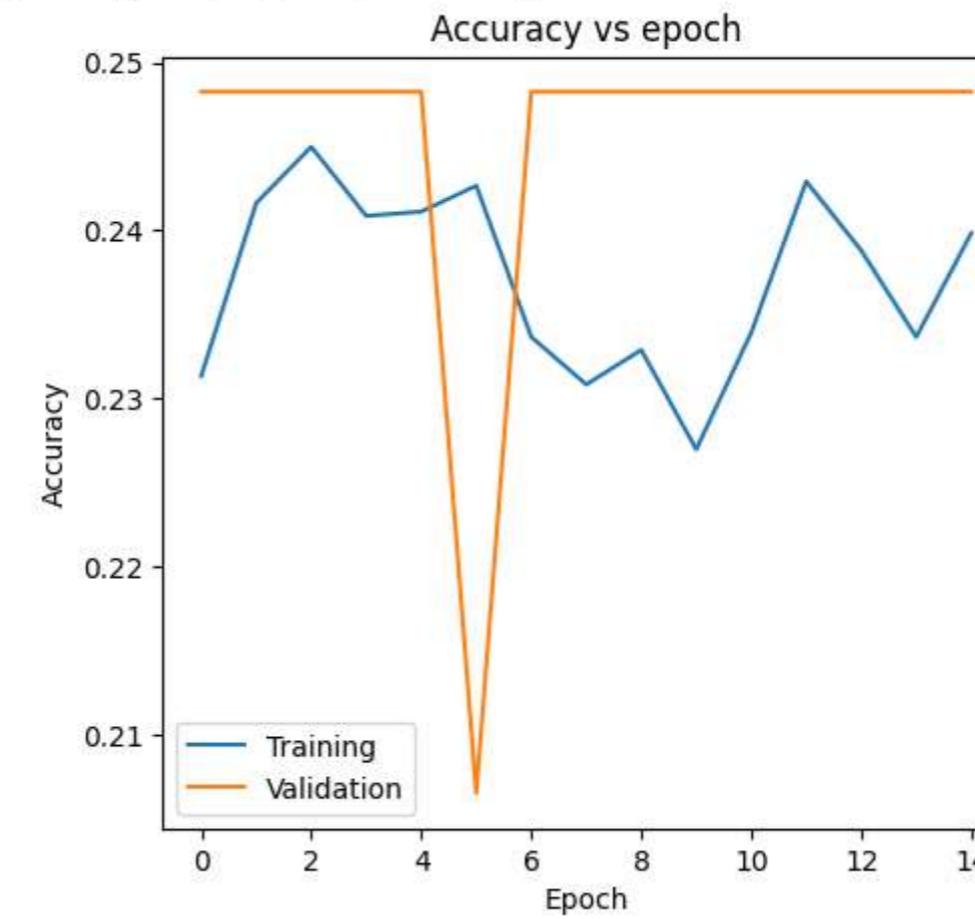
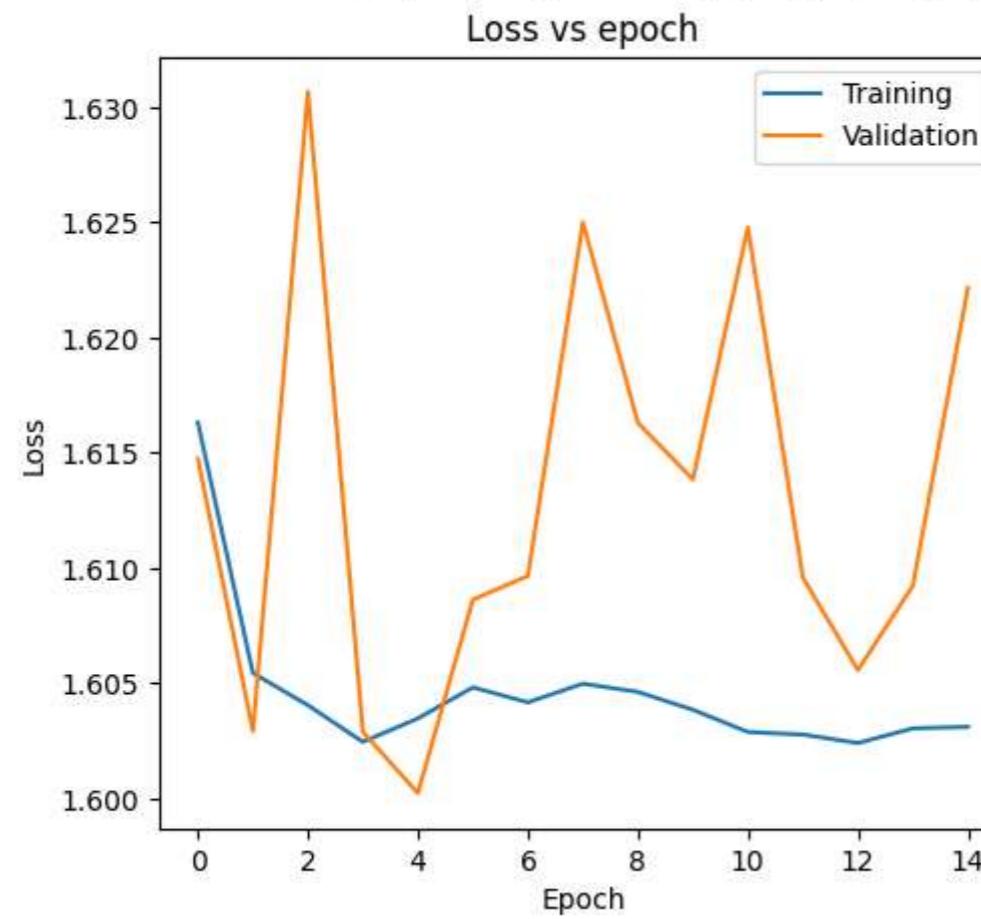
```

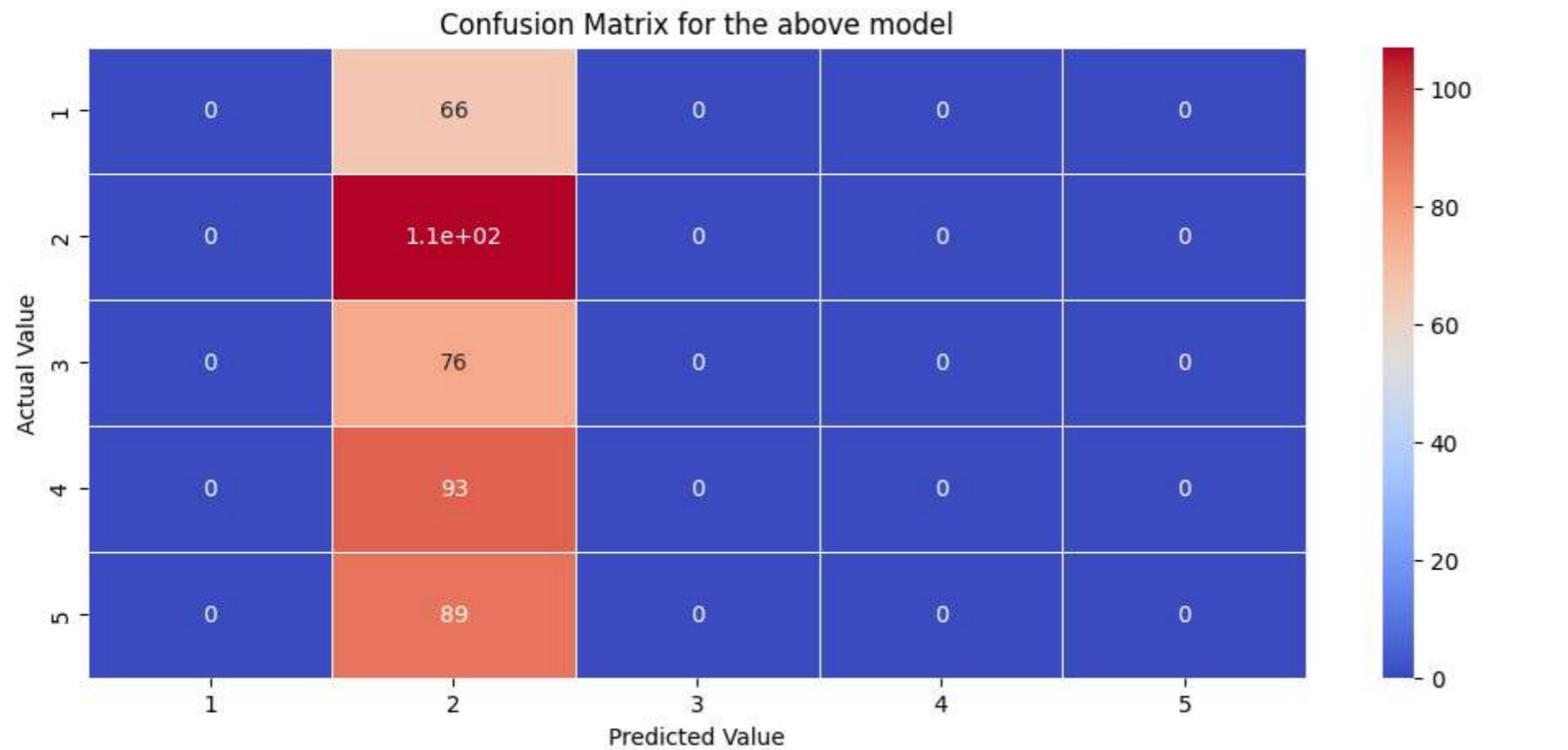
    val_loss, val_acc, val_precision, val_recall = train_model(
        drop_rate=dropout_rate,
        conv_kernels=best_conv_kernel,
        conv_filters=convolution_filters,
        activation=activation,
        pool=best_pool,
        num_fc_layers=best_num_fc_layers,
        train_dataset=train_dataset,
        val_dataset=val_dataset
    )

    result.loc[len(result.index)] = [
        best_conv_kernel,
        convolution_filters,
        best_pool,
        activation,
        best_num_fc_layers,
        dropout_rate,
        val_loss,
        val_acc,
        val_precision,
        val_recall
    ]
]

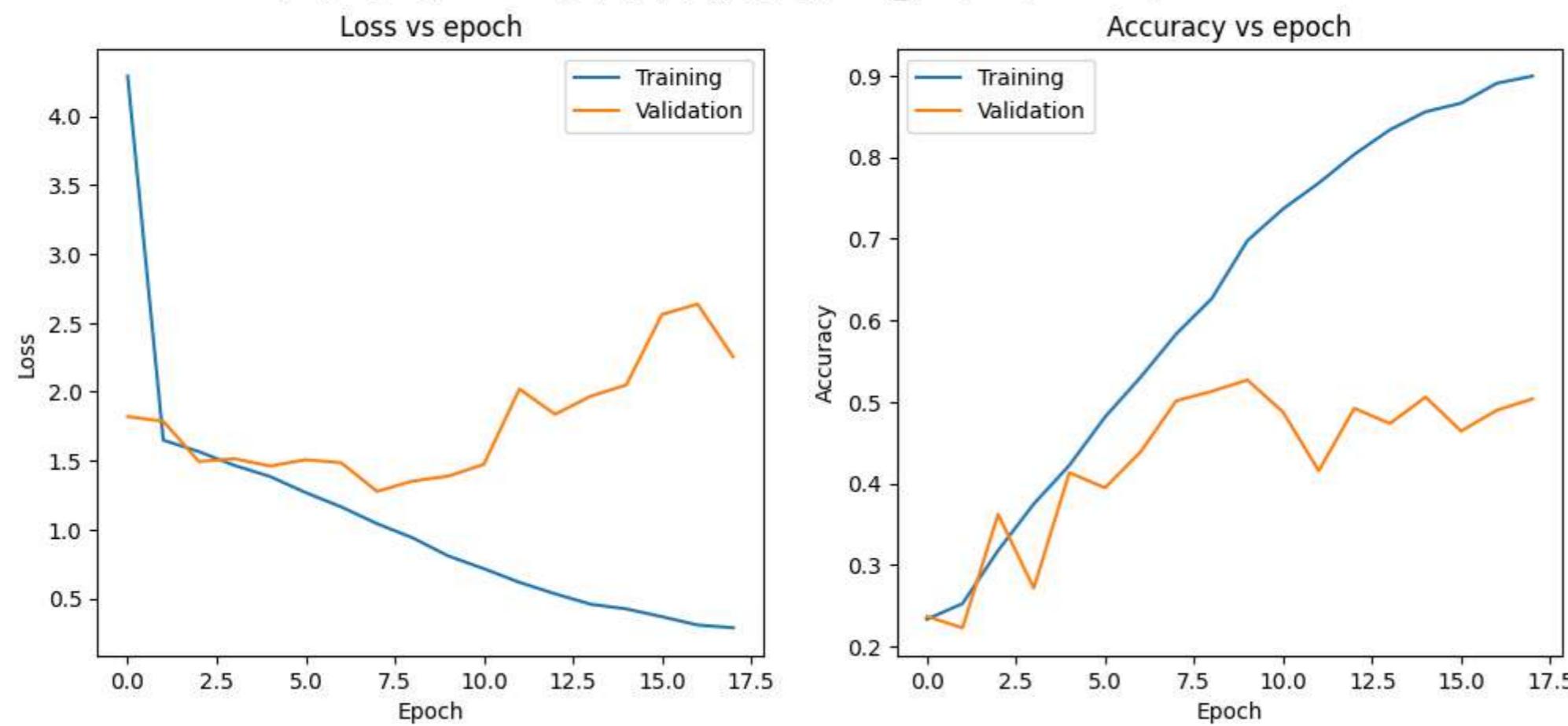
```

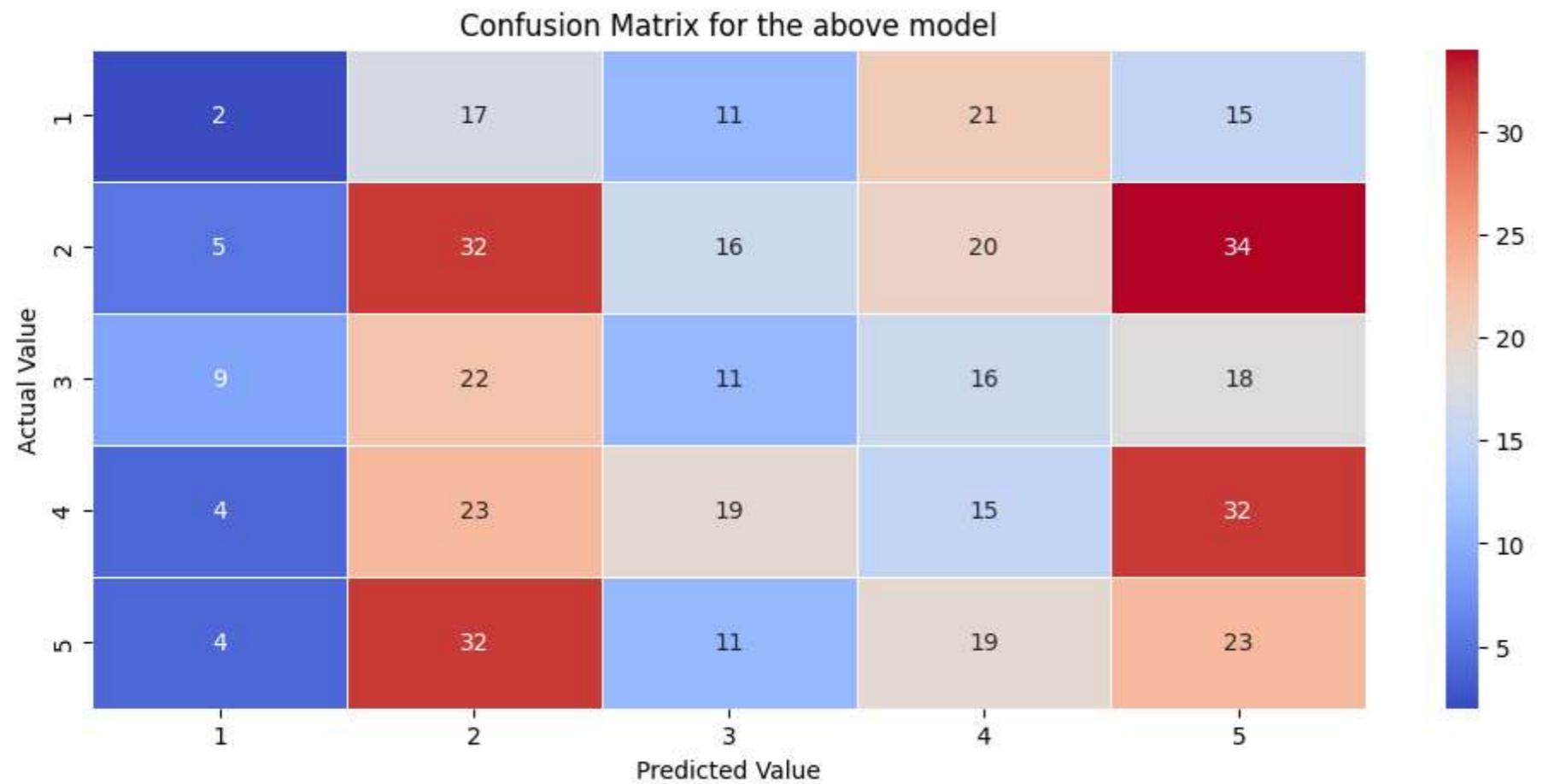
filters [16, 32, 64]; kernels [(3, 3), (3, 3), (5, 5)]; max_pool; sigmoid; dense layers after flatten= 3



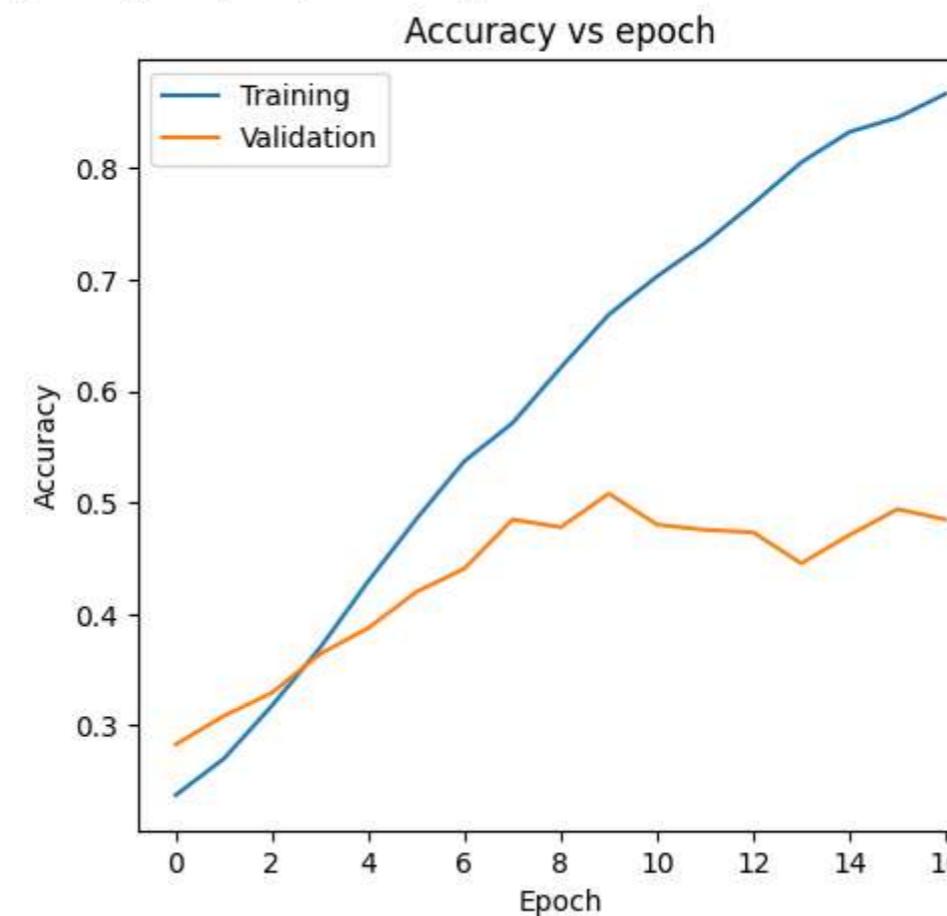
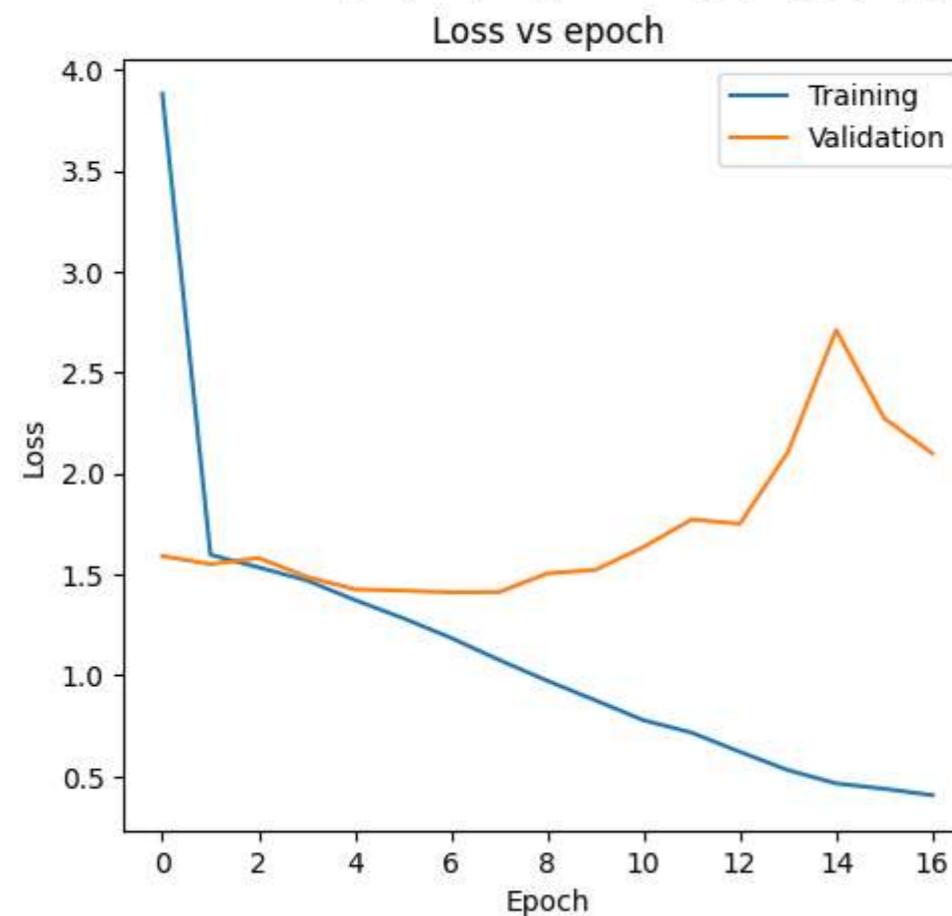


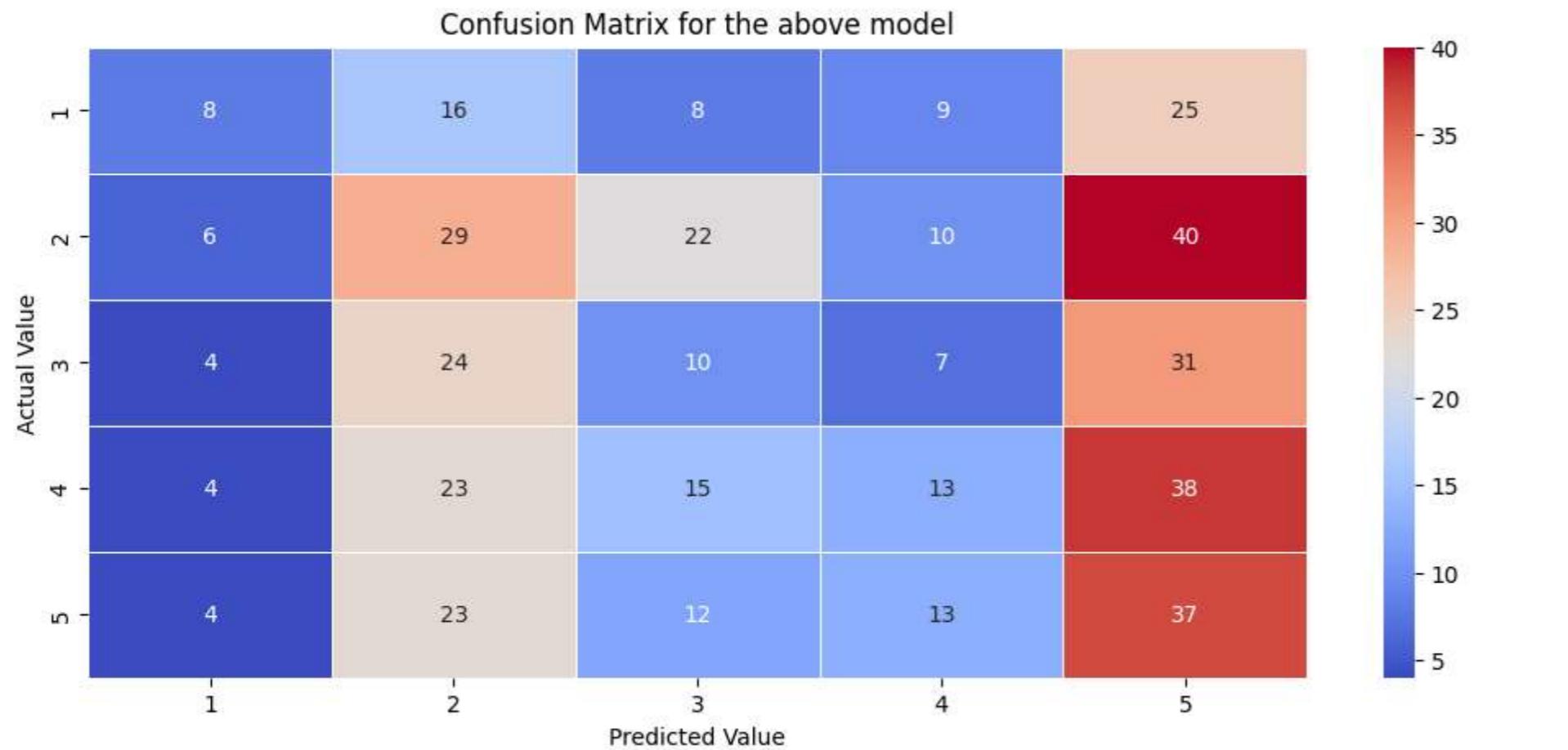
filters [16, 32, 64]; kernels [(3, 3), (3, 3), (5, 5)]; max_pool; elu; dense layers after flatten= 3



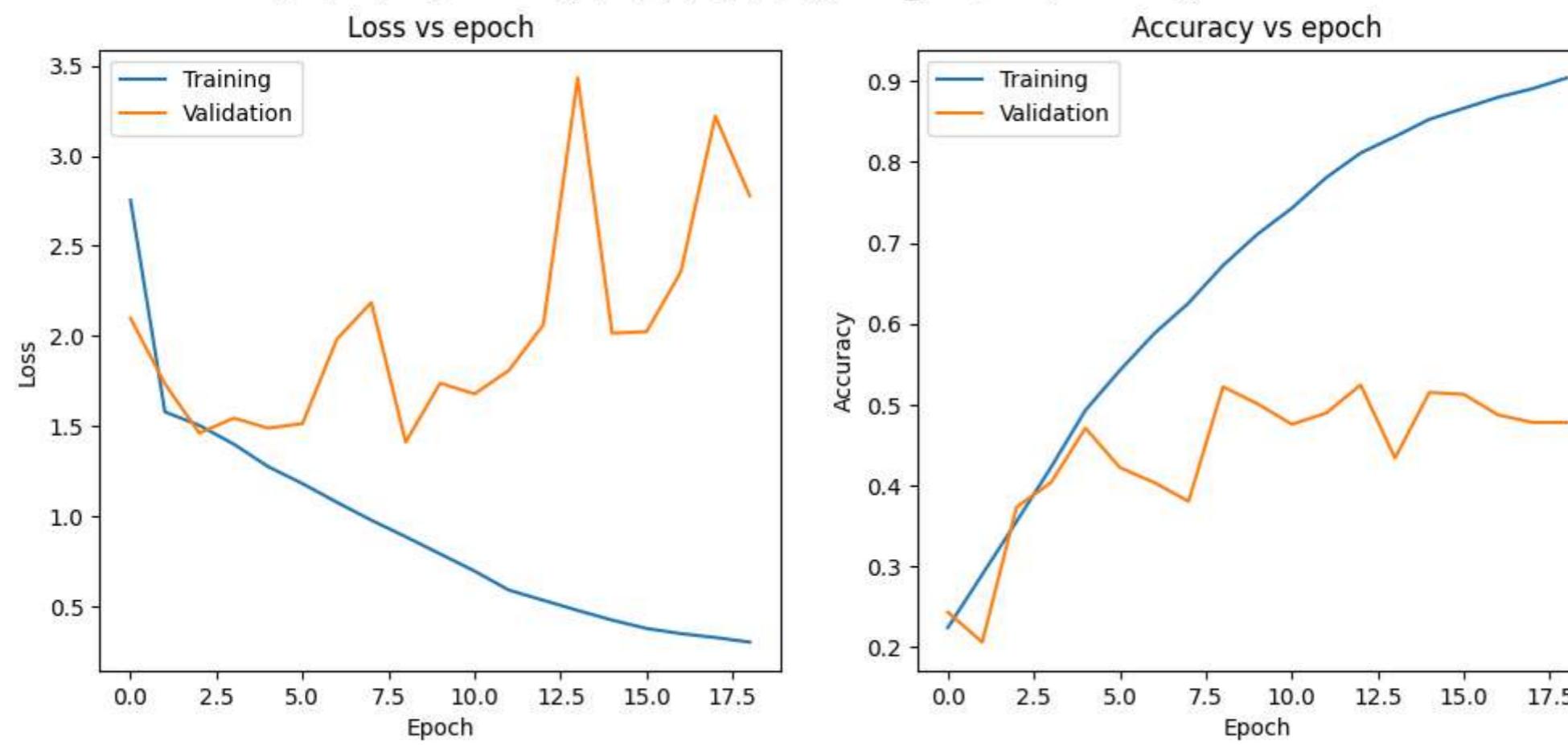


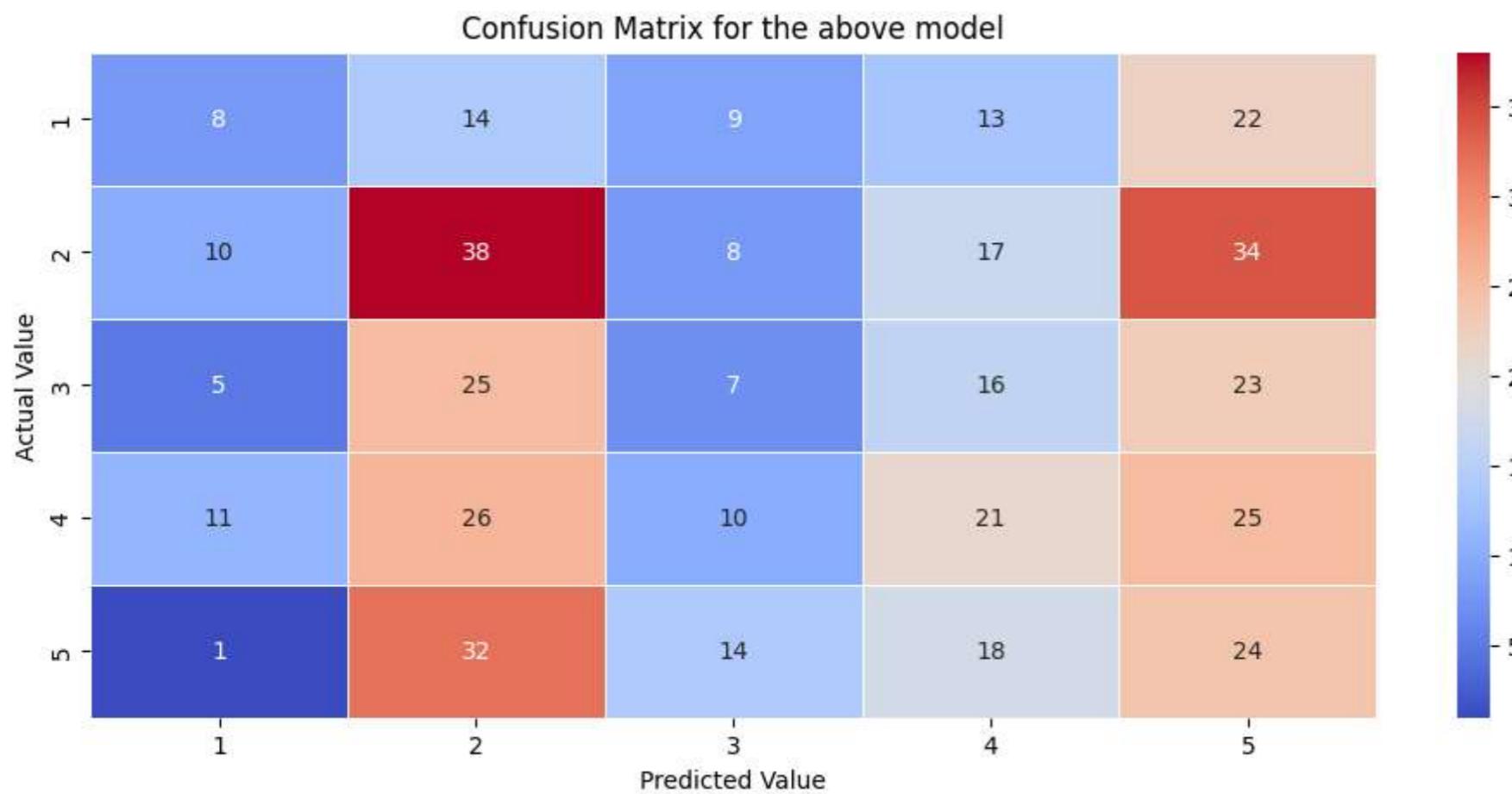
filters [16, 32, 64]; kernels [(3, 3), (3, 3), (5, 5)]; max_pool; relu; dense layers after flatten= 3





filters [16, 32, 64]; kernels [(3, 3), (3, 3), (5, 5)]; max_pool; lrelu; dense layers after flatten= 3



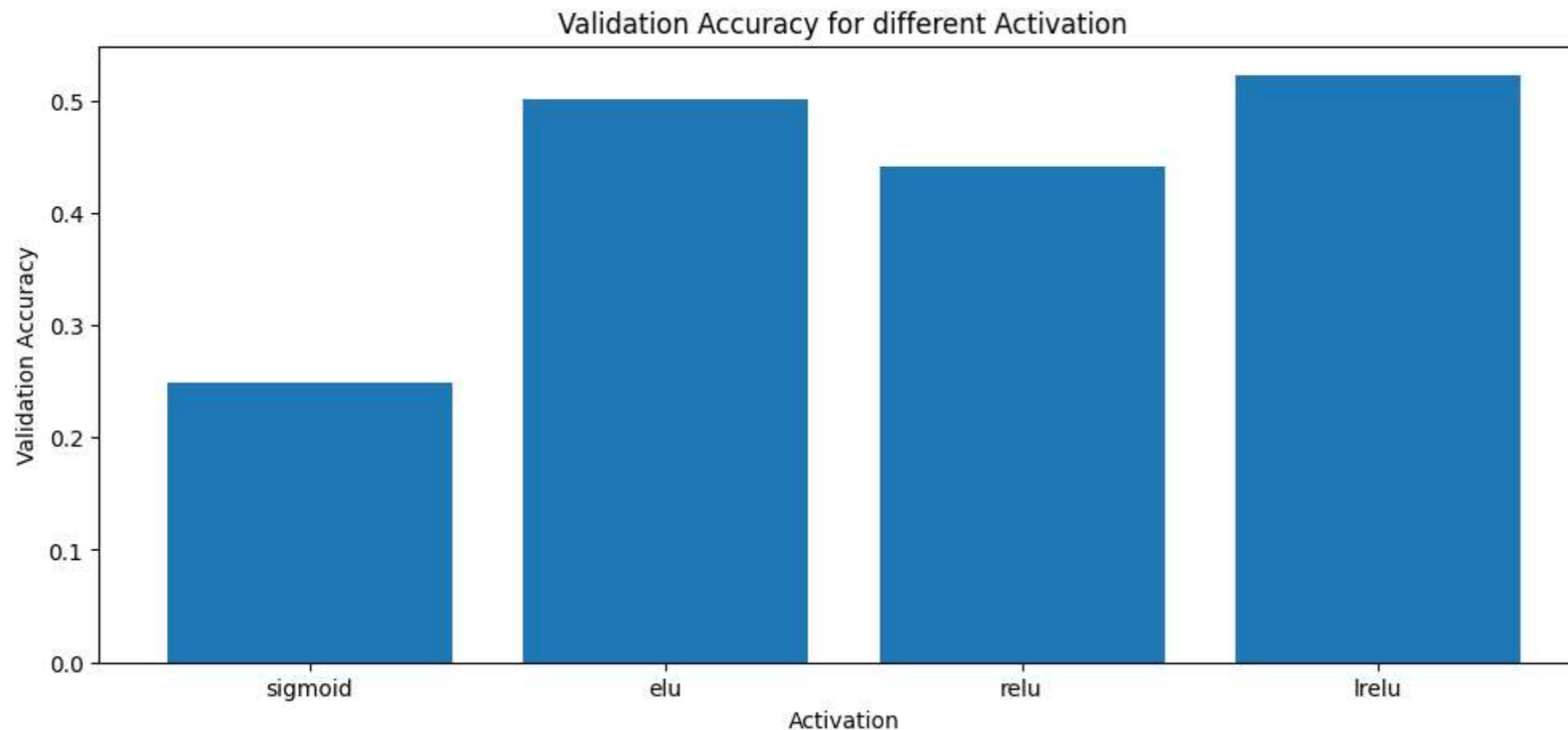


```
In [ ]: result['Validation f1'] = \
    2*result['Validation Precision']*result['Validation Recall']/(result['Validation Precision'] + result['Validation Recall'])
result
```

Out[]:

	Convolution kernel_size	Convolution filters size	Pooling Layers	Activation	FC layer (after Flatten)	Dropout Rate	Validation Loss	Validation Accuracy	Validation Precision	Validation Recall	Validation f1
0	[(3, 3), (3, 3), (5, 5)]	[16, 32, 64]	max	sigmoid		3	0.1	1.600221	0.248260	0.000000	0.000000
1	[(3, 3), (3, 3), (5, 5)]	[16, 32, 64]	max	elu		3	0.1	1.277071	0.501160	0.608108	0.417633
2	[(3, 3), (3, 3), (5, 5)]	[16, 32, 64]	max	relu		3	0.1	1.411410	0.440835	0.576720	0.252900
3	[(3, 3), (3, 3), (5, 5)]	[16, 32, 64]	max	lrelu		3	0.1	1.410656	0.522042	0.595318	0.412993

```
In [ ]: plt.figure(figsize=(12, 5))
plt.bar(
    result['Activation'],
    result['Validation Accuracy'])
plt.ylabel('Validation Accuracy')
plt.xlabel('Activation')
plt.title("Validation Accuracy for different Activation")
plt.show()
```



```
In [ ]: best_activation = result.sort_values(
    by=['Validation Accuracy', 'Validation Loss'],
    ascending=[False, True]
)[['Activation']].iloc[0]

best_activation
```

```
Out[ ]: 'lrelu'
```

Task 4.5

For the best set of parameters from the above runs vary the regularization parameter:

| Regularization ||-----|| Dropout of 0.1 after each layer || Dropout of 0.25 after each layer || Batch Normalization after each layer
(except the first) || Dropout of 0.1 after each layer along with Batch Normalization after each layer (except the first) |

```
In [ ]: result = pd.DataFrame(columns=[
    'Convolution kernel_size',
    'Convolution filters size',
    'Pooling Layers',
    'Activation',
    'FC layer (after Flatten)',
    'Dropout Rate',
    'Batch Normalization Present',
    'Validation Loss',
    'Validation Accuracy',
    'Validation Precision',
    'Validation Recall'
])
```

```

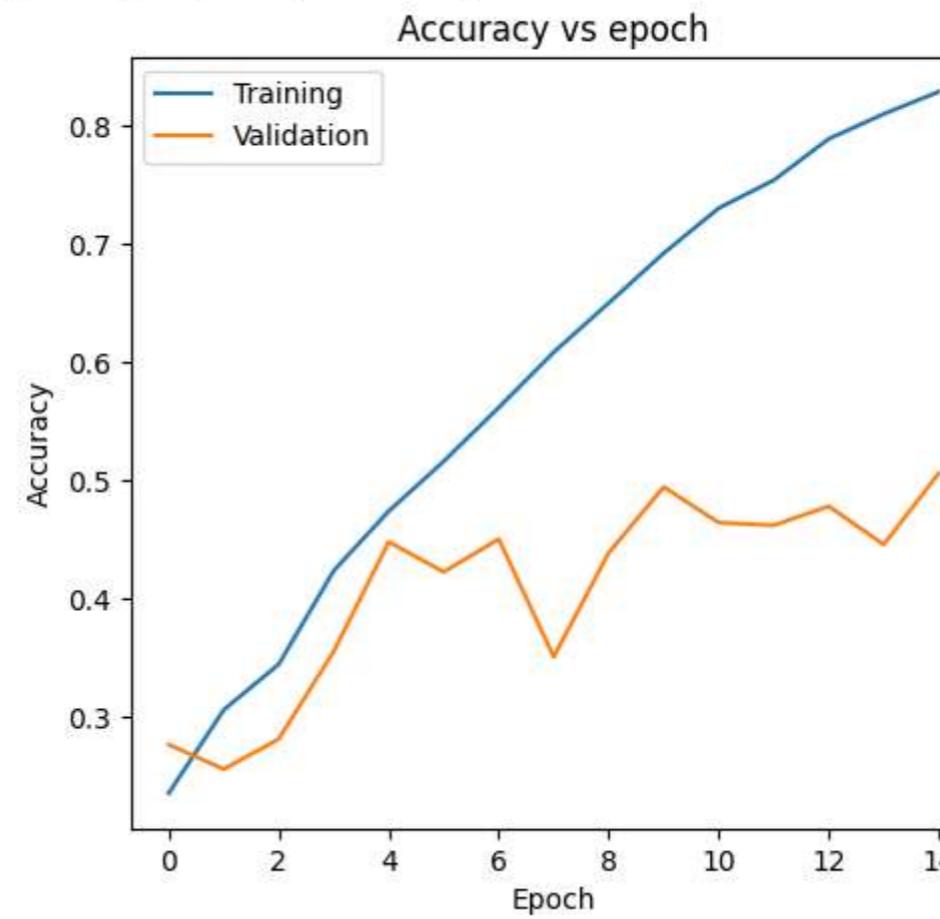
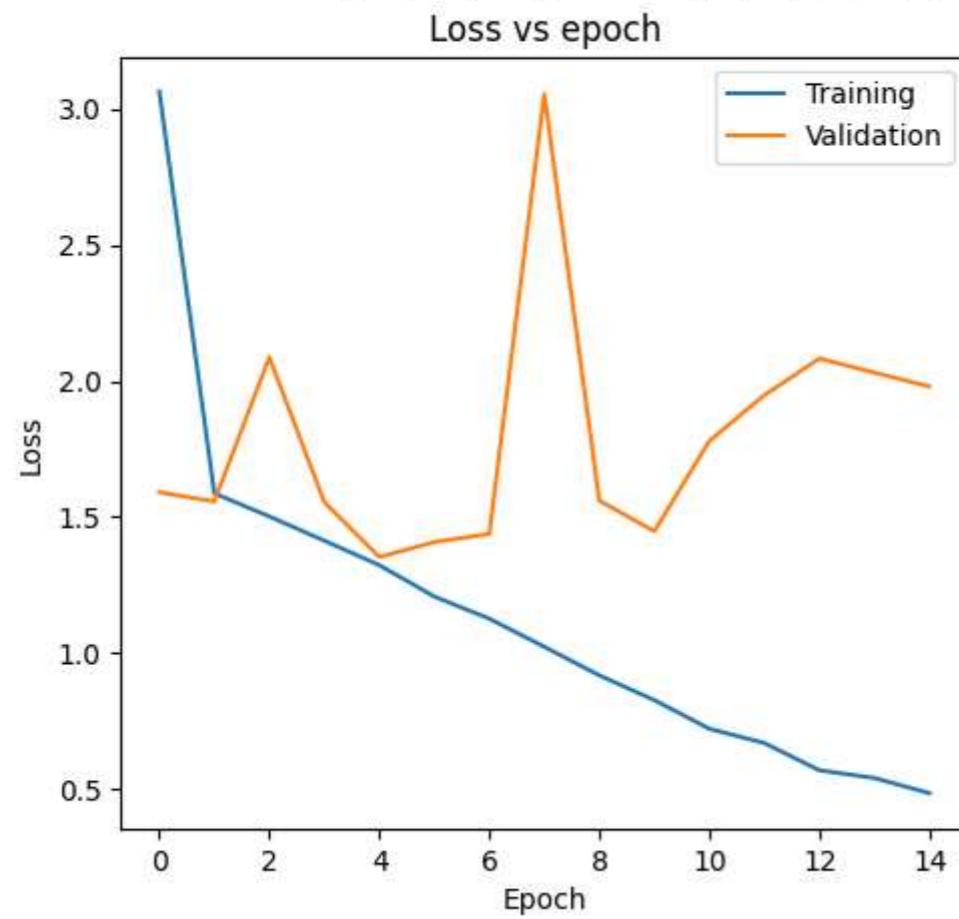
convolution_filters = [16, 32, 64]
dropout_rates = [0.1, 0.25, 0, 0.1]
batch_norm_cases = [False, False, True, True]

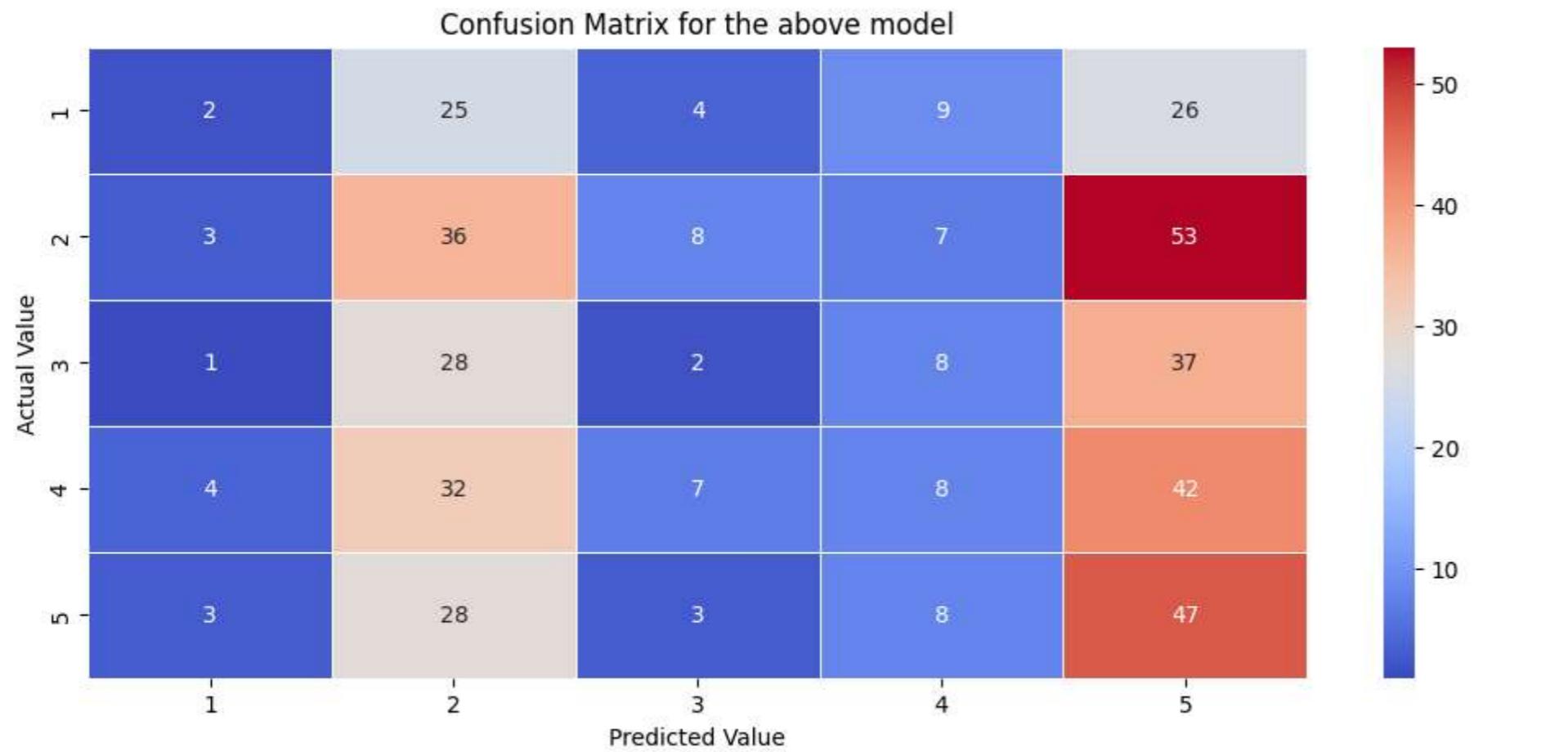
for dropout_rate, do_batch in zip(dropout_rates, batch_norm_cases):
    val_loss, val_acc, val_precision, val_recall = train_model(
        drop_rate=dropout_rate,
        conv_kernels=best_conv_kernel,
        conv_filters=convolution_filters,
        activation=best_activation,
        pool=best_pool,
        add_batch_norm=do_batch,
        num_fc_layers=best_num_fc_layers,
        train_dataset=train_dataset,
        val_dataset=val_dataset
    )

    result.loc[len(result.index)] = [
        best_conv_kernel,
        convolution_filters,
        best_pool,
        best_activation,
        best_num_fc_layers,
        dropout_rate,
        do_batch,
        val_loss,
        val_acc,
        val_precision,
        val_recall
    ]
]

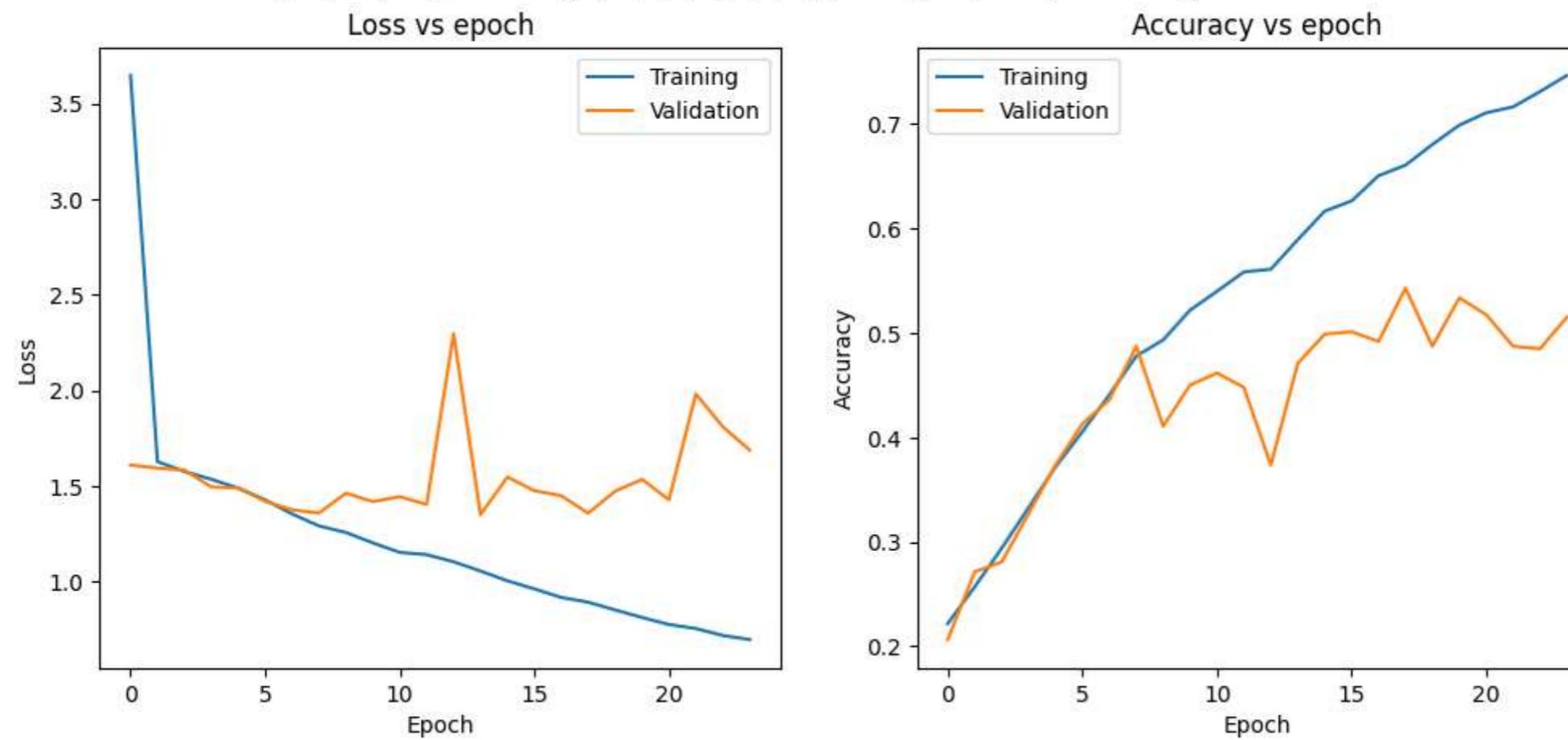
```

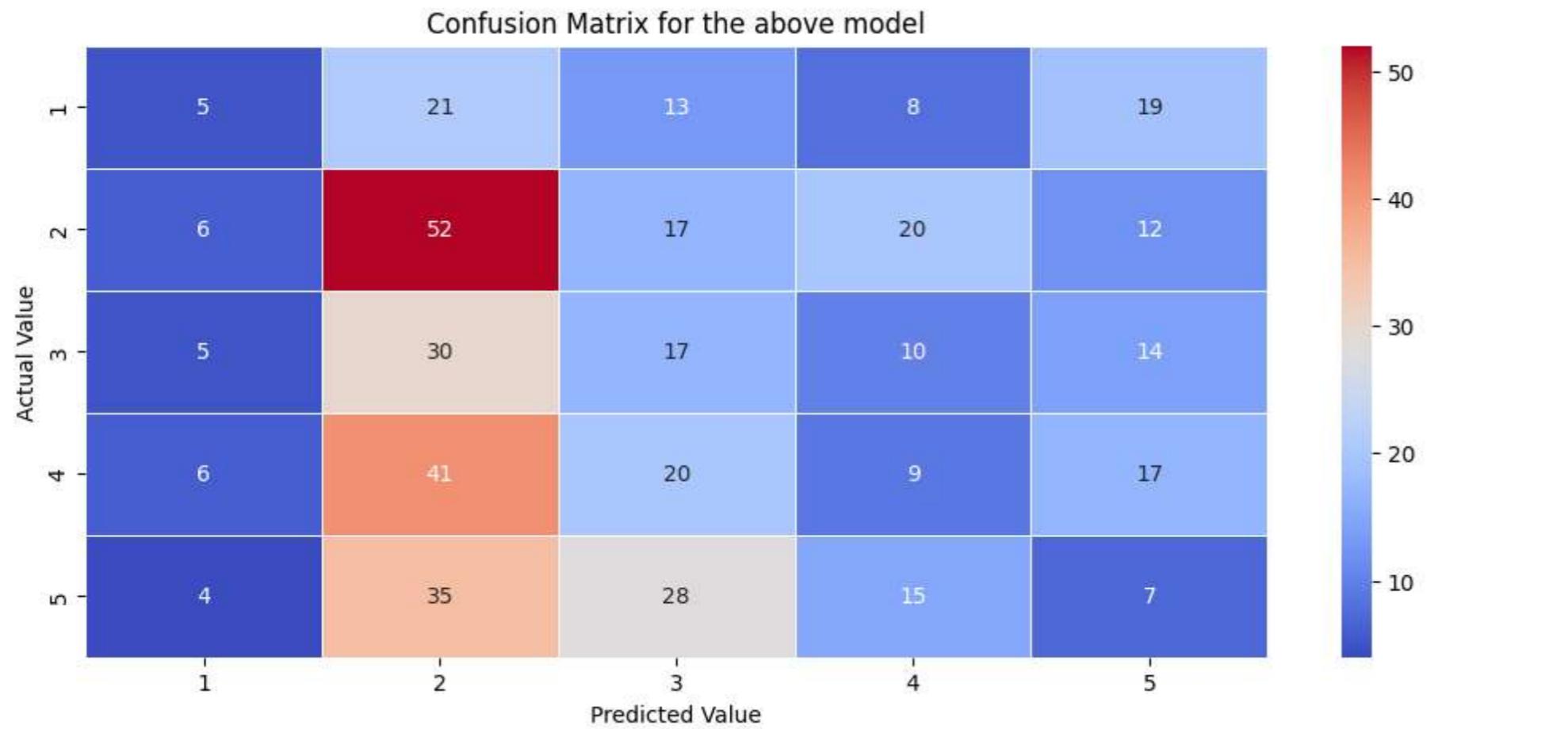
filters [16, 32, 64]; kernels [(3, 3), (3, 3), (5, 5)]; max_pool; lrelu; dense layers after flatten= 3



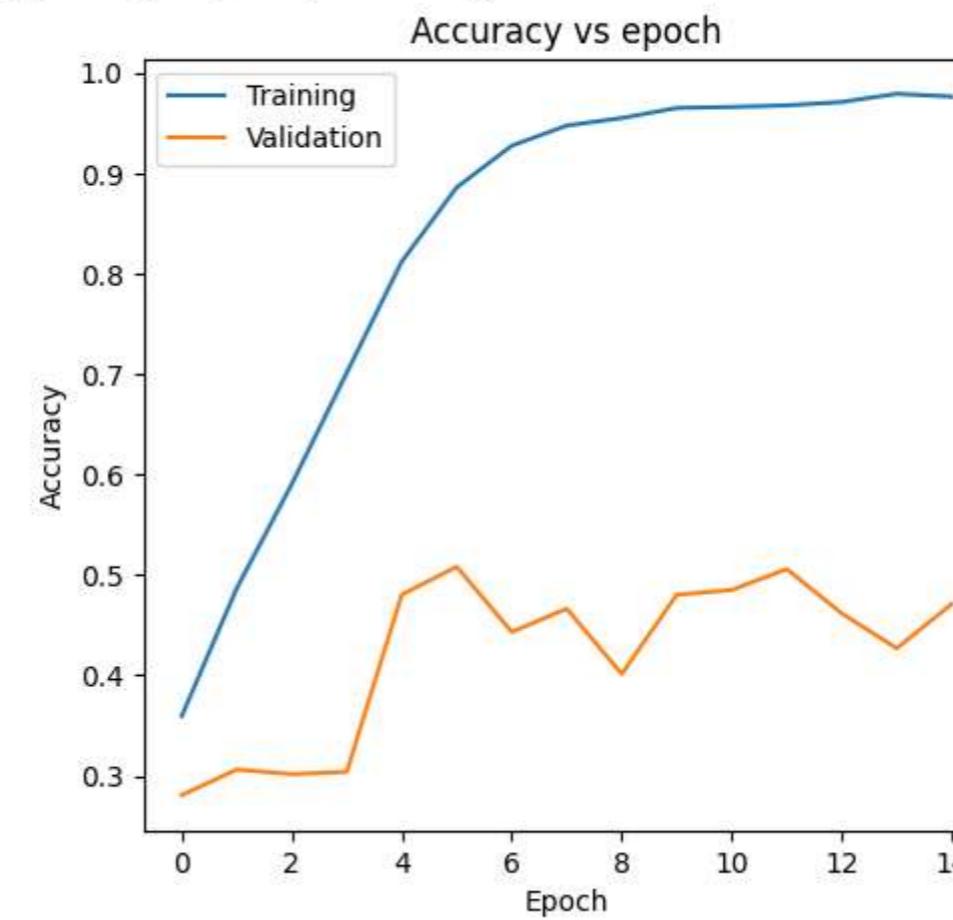
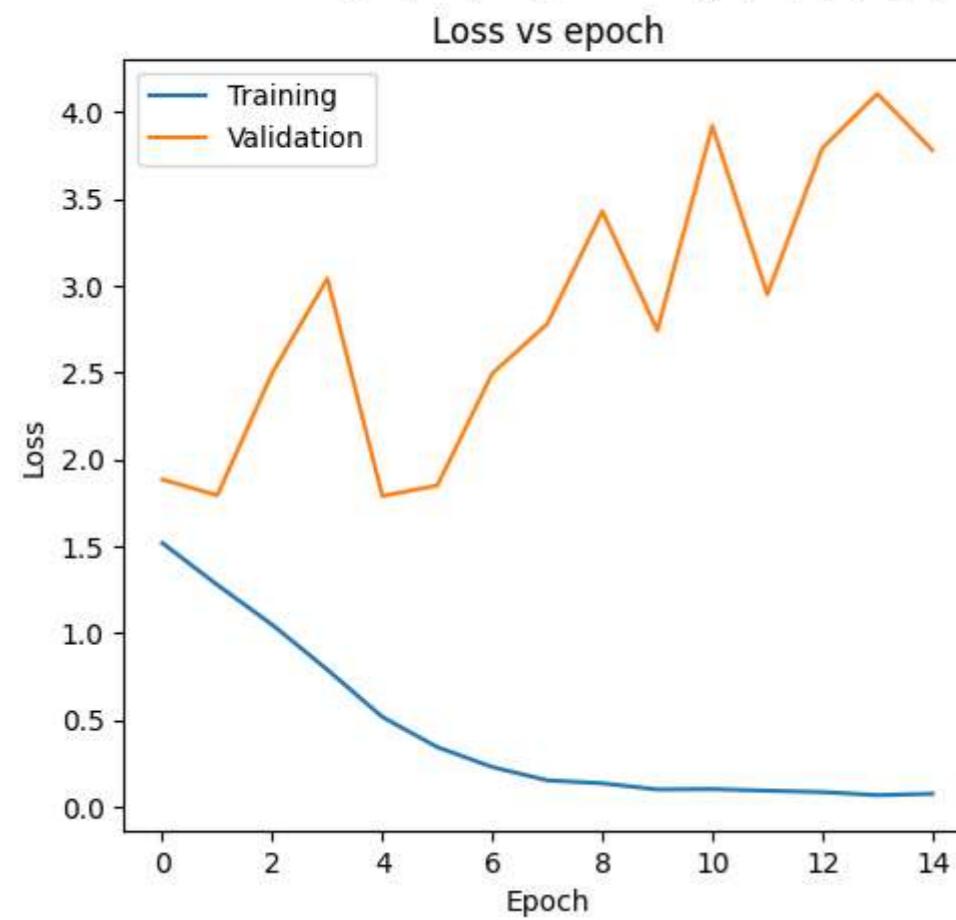


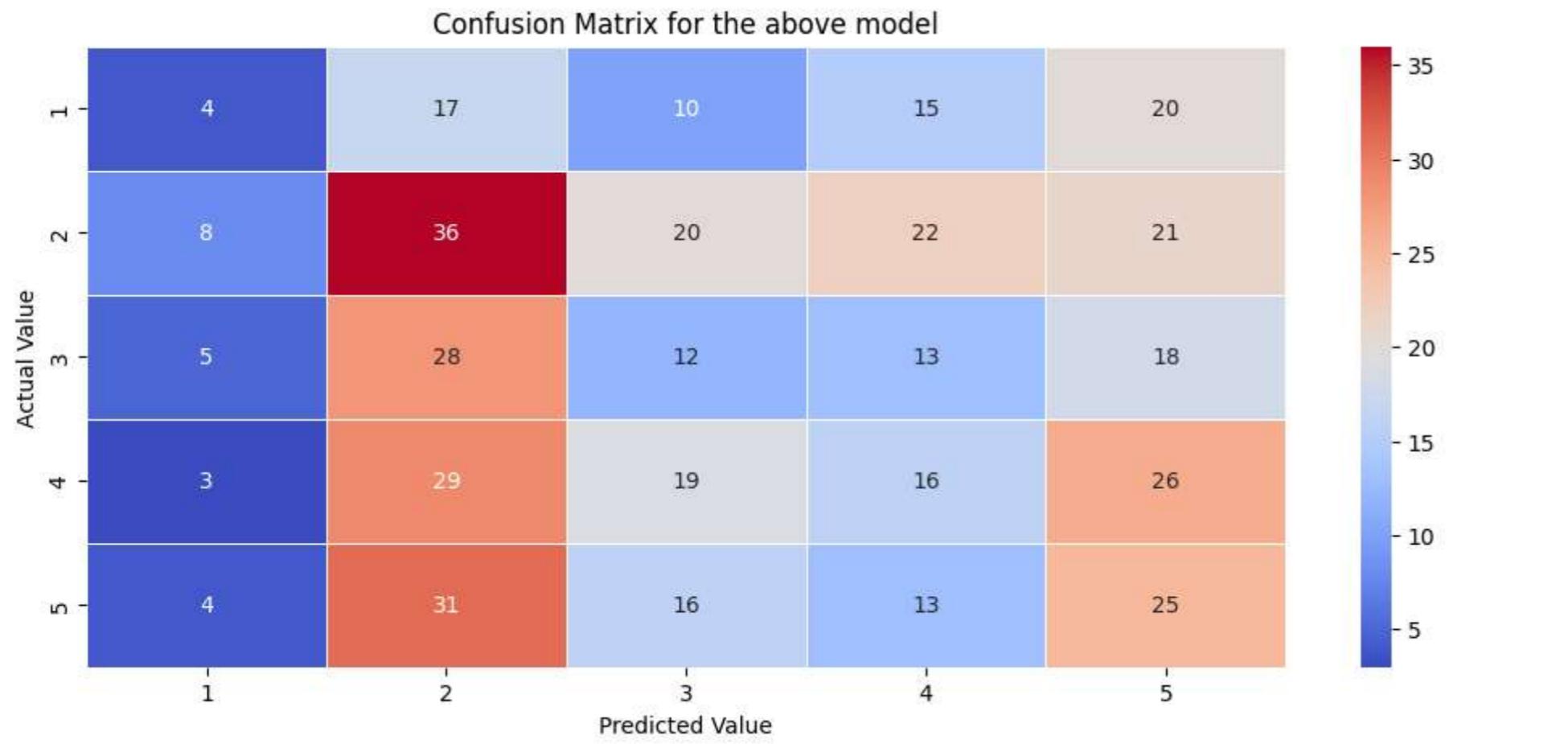
filters [16, 32, 64]; kernels [(3, 3), (3, 3), (5, 5)]; max_pool; lrelu; dense layers after flatten= 3



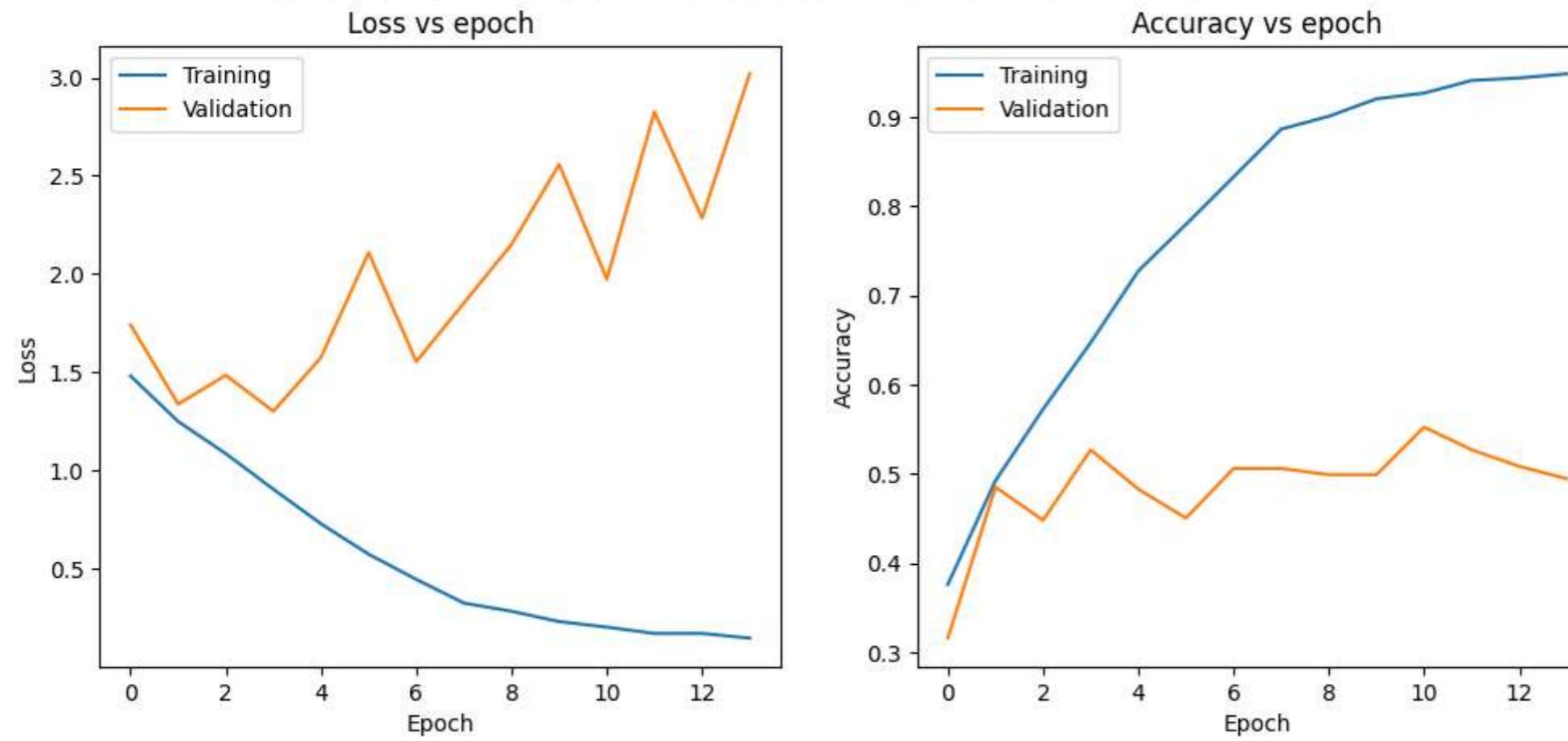


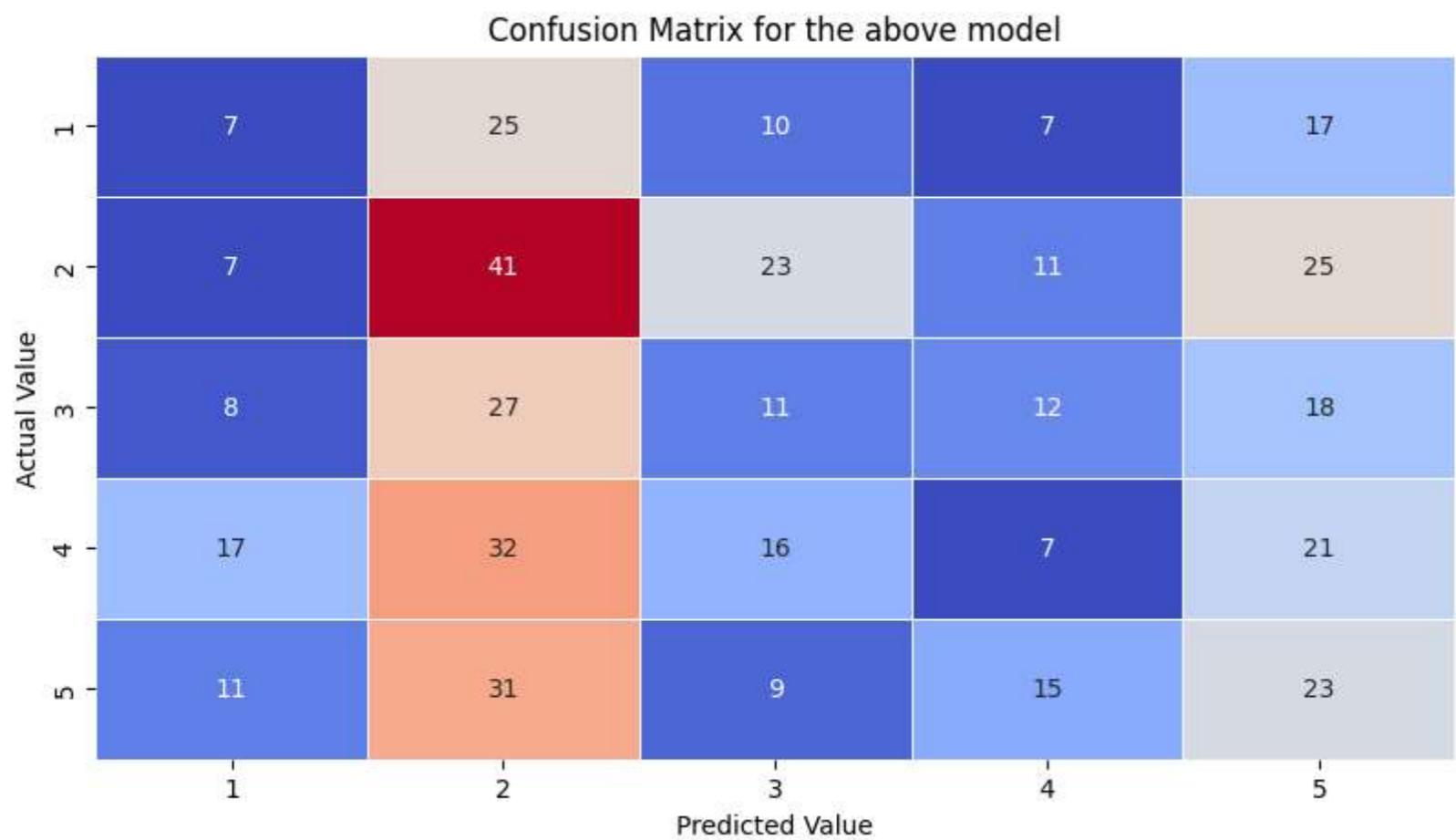
filters [16, 32, 64]; kernels [(3, 3), (3, 3), (5, 5)]; max_pool; lrelu; dense layers after flatten= 3





filters [16, 32, 64]; kernels [(3, 3), (3, 3), (5, 5)]; max_pool; lrelu; dense layers after flatten= 3



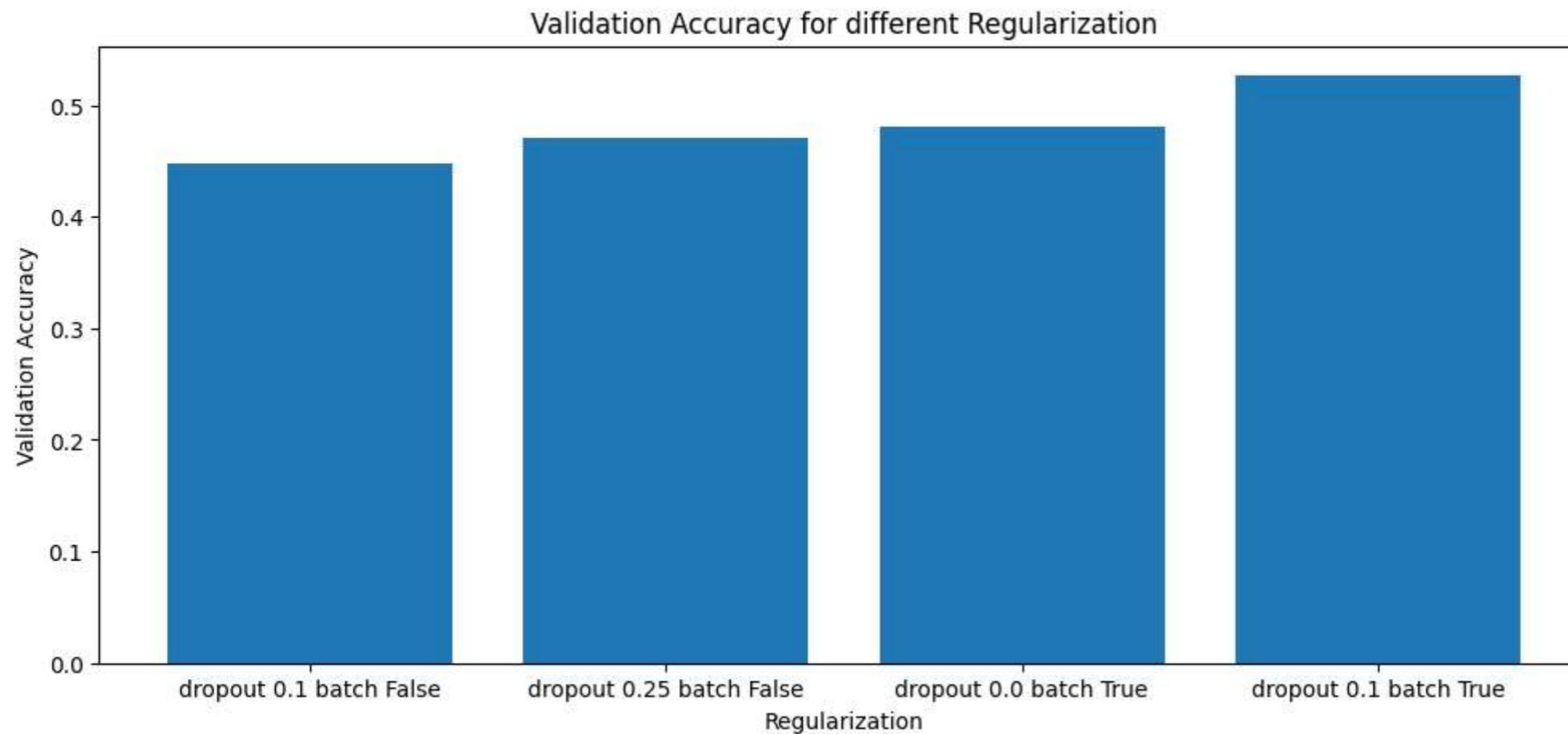


```
In [ ]: result['Validation f1'] = \
    2*result['Validation Precision']*result['Validation Recall']/(result['Validation Precision'] + result['Validation Recall'])
result
```

Out[]:

	Convolution kernel_size	Convolution filters size	Pooling Layers	Activation	FC layer (after Flatten)	Dropout Rate	Batch Normalization Present	Validation Loss	Validation Accuracy	Validation Precision	Validation Recall	Validation f1
0	[(3, 3), (3, 3), (5, 5)]	[16, 32, 64]	max	lrelu	3	0.10	False	1.353482	0.447796	0.551136	0.225058	0.319605
1	[(3, 3), (3, 3), (5, 5)]	[16, 32, 64]	max	lrelu	3	0.25	False	1.349767	0.470998	0.610837	0.287703	0.391167
2	[(3, 3), (3, 3), (5, 5)]	[16, 32, 64]	max	lrelu	3	0.00	True	1.788964	0.480278	0.510417	0.454756	0.480982
3	[(3, 3), (3, 3), (5, 5)]	[16, 32, 64]	max	lrelu	3	0.10	True	1.300549	0.526682	0.580645	0.417633	0.485830

```
In [ ]: plt.figure(figsize=(12, 5))
plt.bar(
    [f'dropout {drop} batch {batch}' for drop,batch in zip(result['Dropout Rate'], result['Batch Normalization Present'])],
    result['Validation Accuracy'])
plt.ylabel('Validation Accuracy')
plt.xlabel('Regularization')
plt.title("Validation Accuracy for different Regularization")
plt.show()
```



```
In [ ]: best_dropout = result.sort_values(
    by=['Validation Accuracy', 'Validation Loss'],
    ascending=[False, True]
)[['Dropout Rate']].iloc[0]

print(f"best dropout rate: {best_dropout}")

best_do_batch = result.sort_values(
    by=['Validation Accuracy', 'Validation Loss'],
    ascending=[False, True]
)[['Batch Normalization Present']].iloc[0]

print(f"Batch Normalization present for best case: {best_do_batch}")

best dropout rate: 0.1
Batch Normalization present for best case: True
```

Task 4.6

For the best set of parameters from the above runs, add [1,2,3] more convolution layers, and compare the size of trainable parameters and also compare the time required to train each model for 10 epoch.

```
In [ ]: import time

result = pd.DataFrame(columns=[
    'Convolution kernel_size',
    'Convolution filters size',
    'Pooling Layers',
    'Activation',
    'FC layer (after Flatten)',
    'Dropout Rate',
    'Batch Normalization Present',
```

```
'Extra Conv Layers',
'Validation Loss',
'Validation Accuracy',
'Validation Precision',
'Validation Recall',
'Time Taken'
])

convolution_filters = [16, 32, 64]
extra_conv_layers = [0, 1, 2, 3]

for extra_conv_layer in extra_conv_layers:
    # making a deep copy to avoid editing of original element
    convolution_filters_copy = []
    best_conv_kernel_copy = []

    for filtr, kernel in zip(convolution_filters, best_conv_kernel):
        convolution_filters_copy.append(filtr)
        best_conv_kernel_copy.append(kernel)

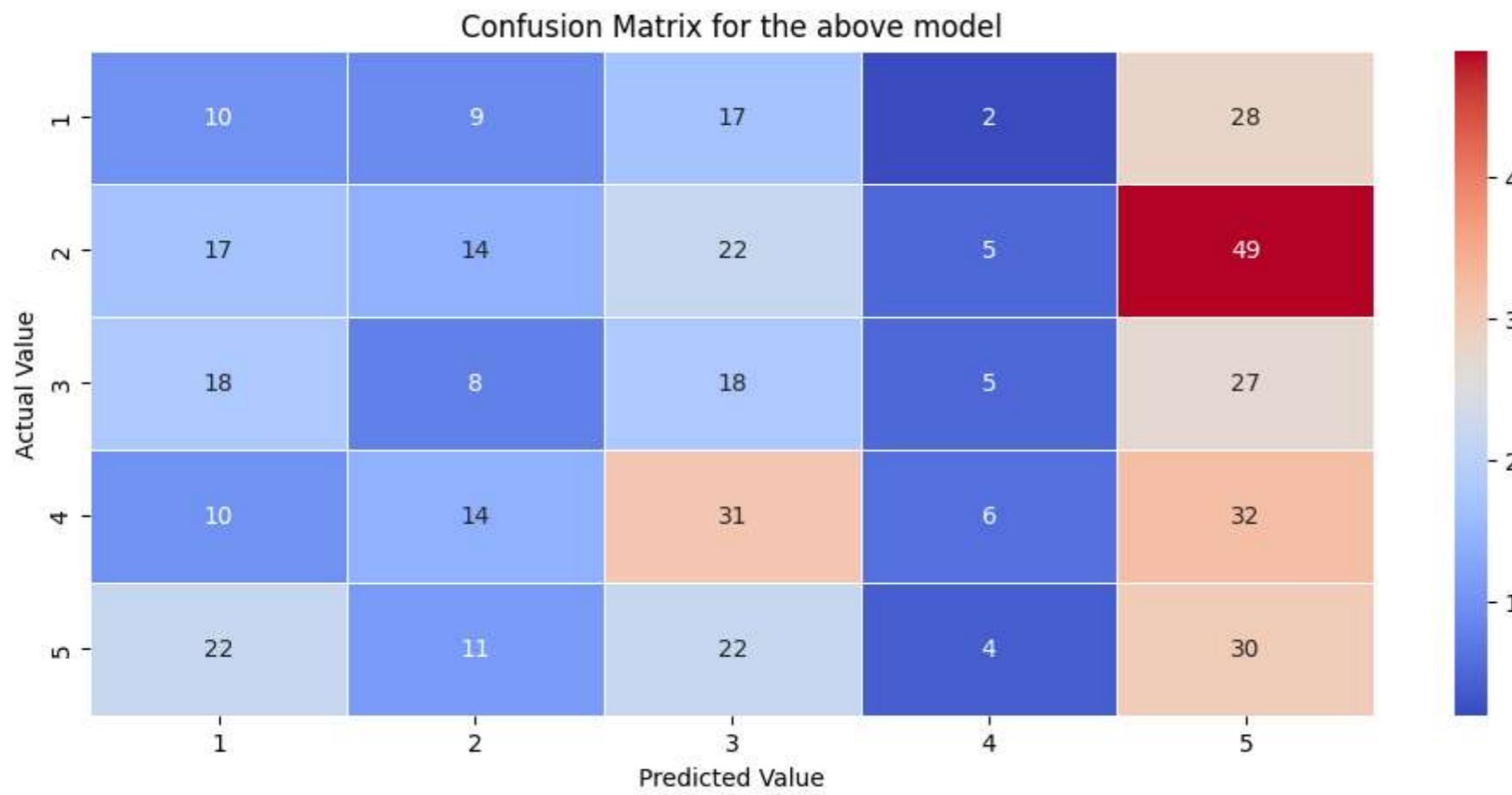
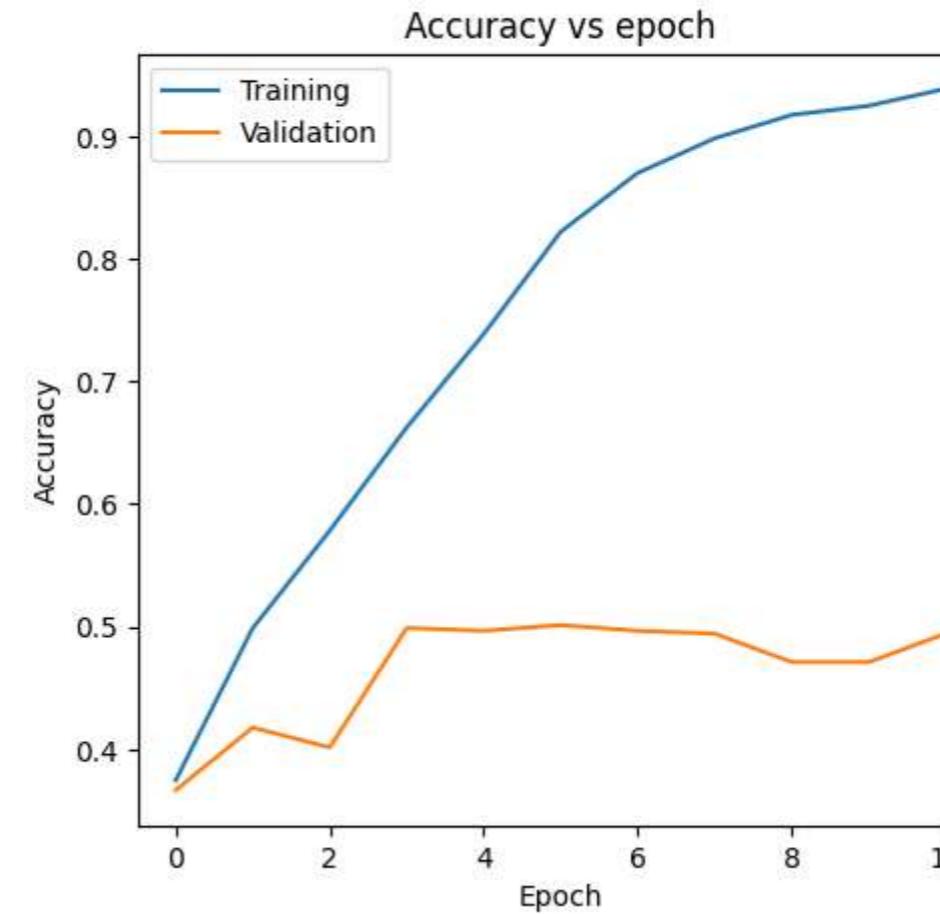
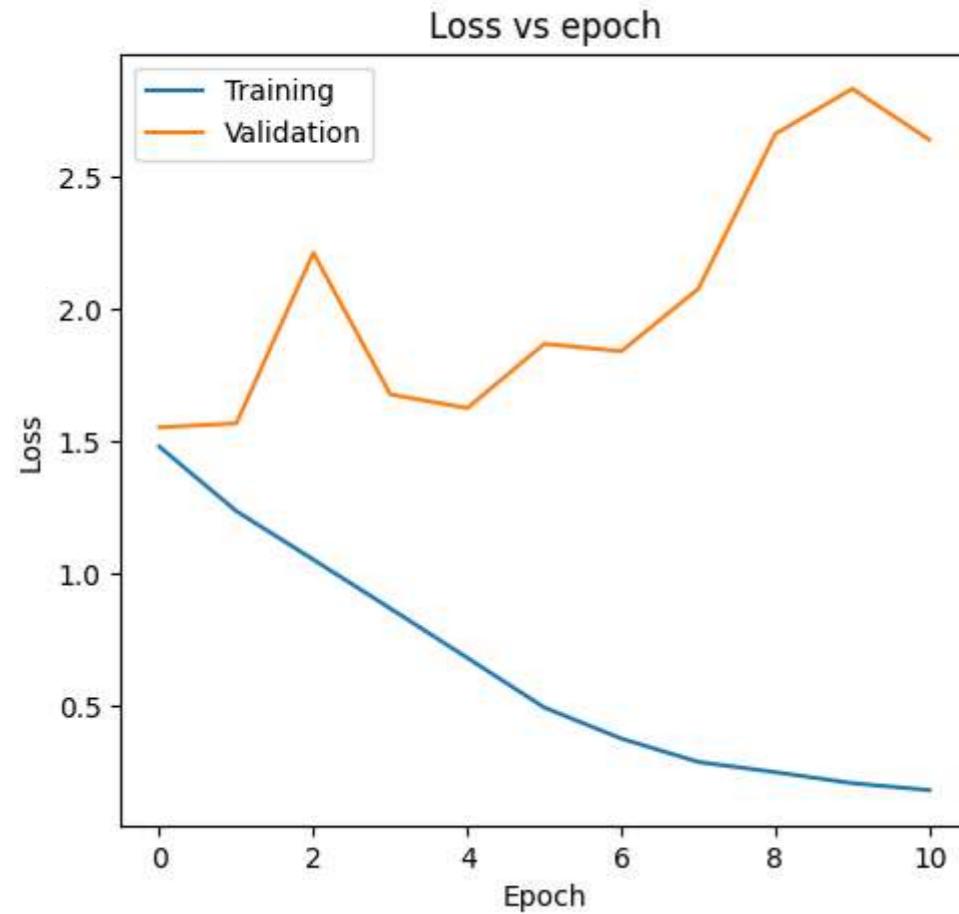
    start_time = time.time()

    val_loss, val_acc, val_precision, val_recall = train_model(
        drop_rate=best_dropout,
        conv_kernels=best_conv_kernel_copy,
        conv_filters=convolution_filters_copy,
        activation=best_activation,
        pool=best_pool,
        add_batch_norm=best_do_batch,
        num_fc_layers=best_num_fc_layers,
        extra_conv_layers=extra_conv_layer,
        train_dataset=train_dataset,
        val_dataset=val_dataset,
    )
    end_time = time.time()

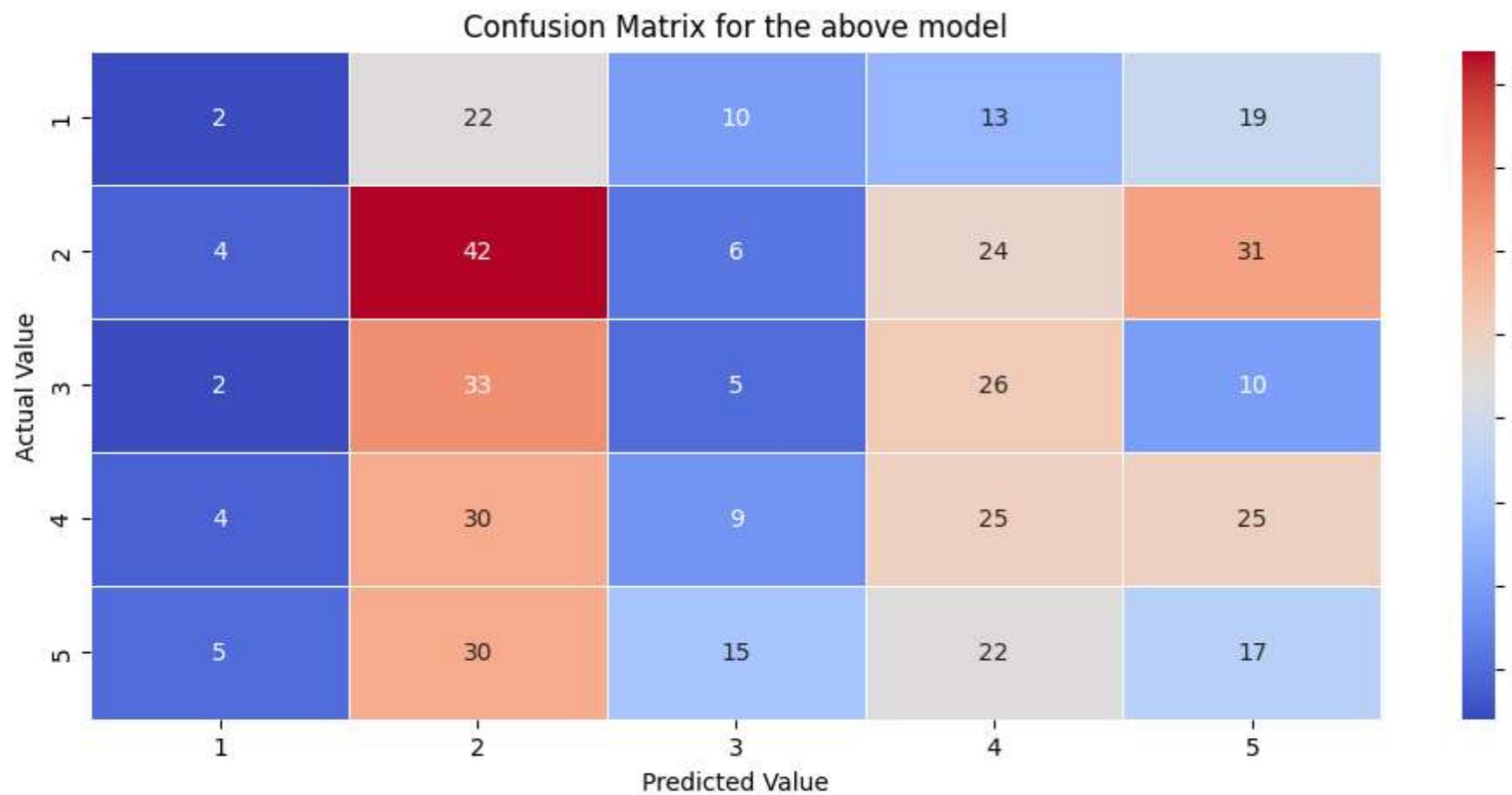
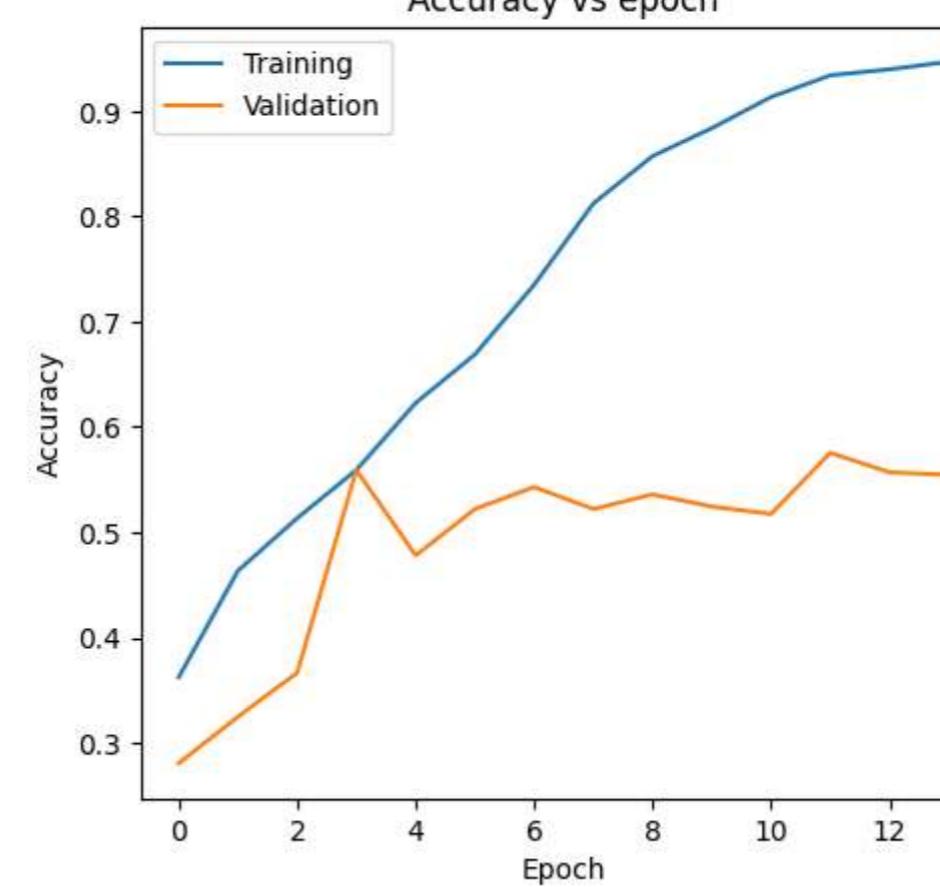
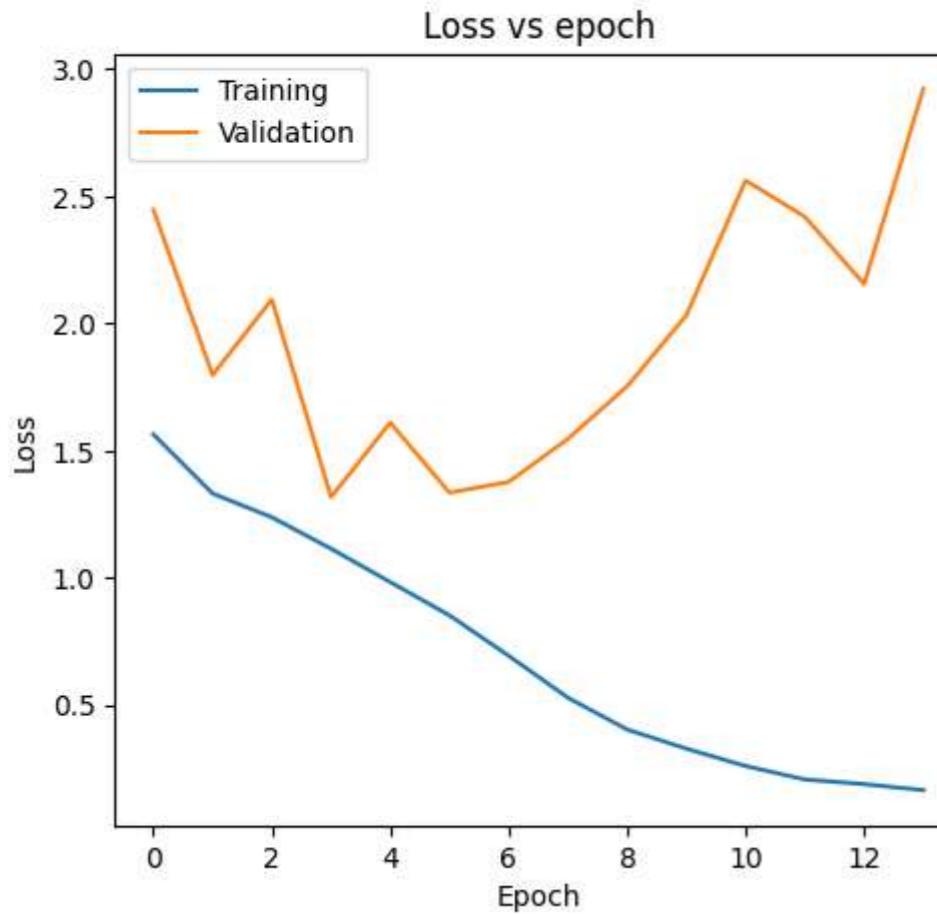
    time_taken = end_time - start_time

    result.loc[len(result.index)] = [
        best_conv_kernel_copy,
        convolution_filters_copy,
        best_pool,
        best_activation,
        best_num_fc_layers,
        best_dropout,
        best_do_batch,
        extra_conv_layer,
        val_loss,
        val_acc,
        val_precision,
        val_recall,
        time_taken
    ]
```

filters [16, 32, 64]; kernels [(3, 3), (3, 3), (5, 5)]; max_pool; lrelu; dense layers after flatten= 3

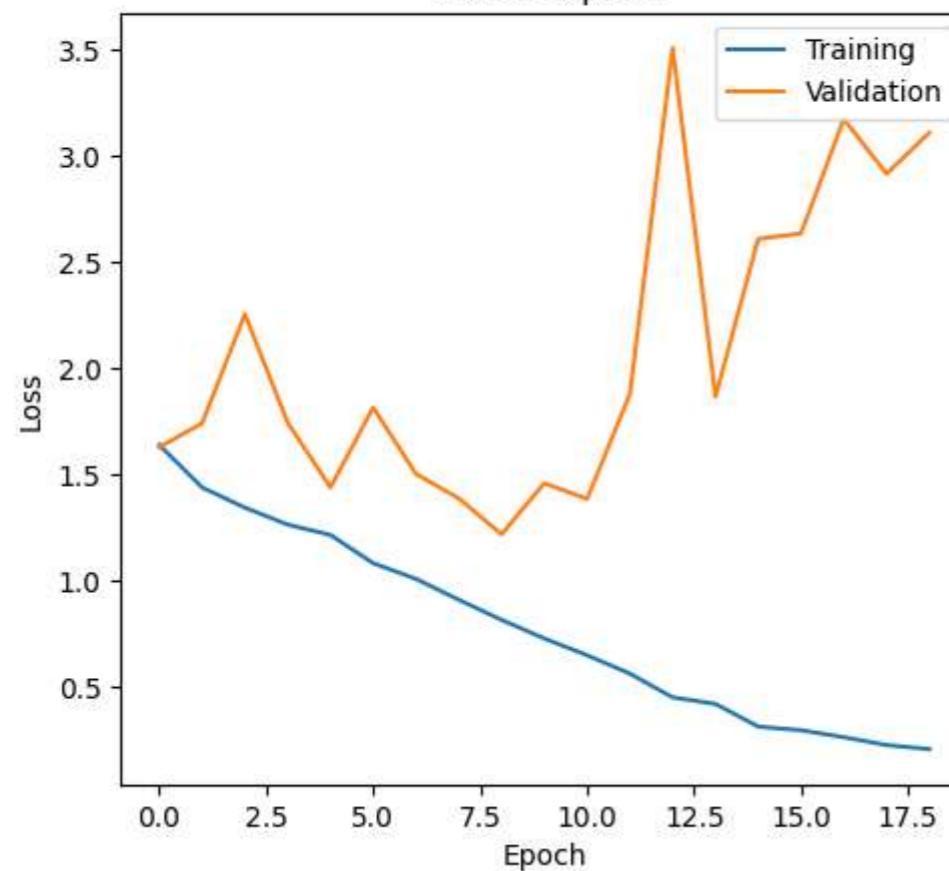


filters [16, 32, 64, 128]; kernels [(3, 3), (3, 3), (5, 5), (5, 5)]; max_pool; lrelu; dense layers after flatten= 3

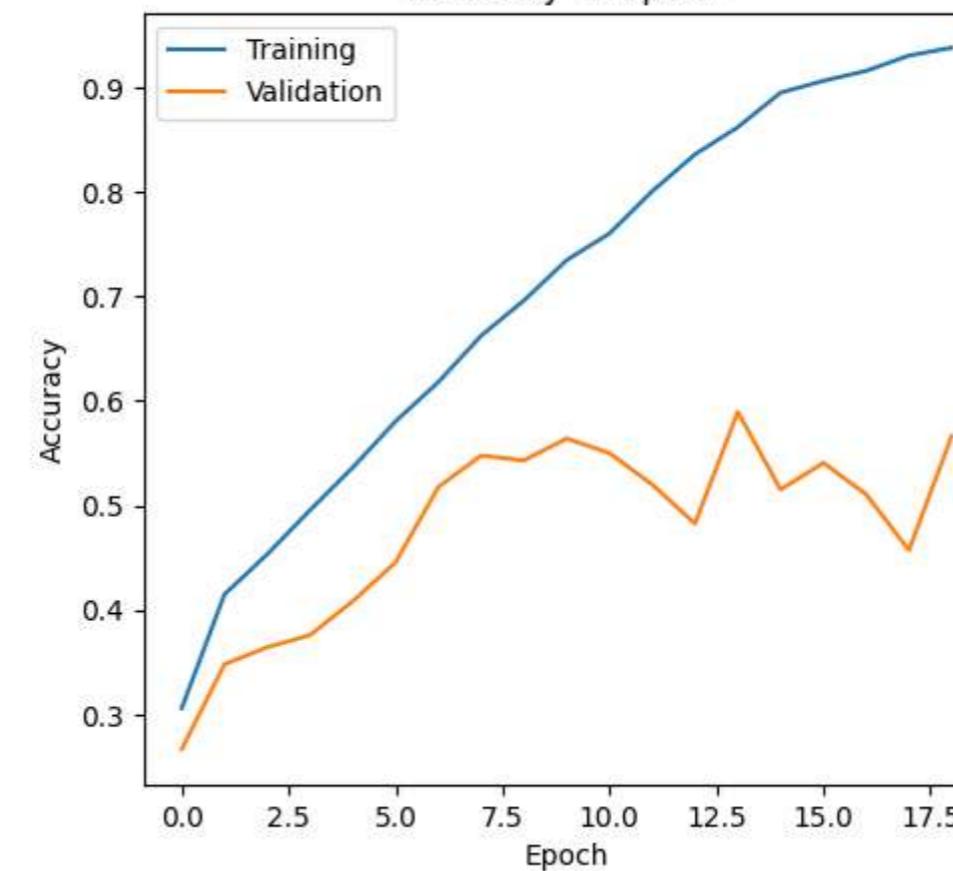


filters [16, 32, 64, 128, 256]; kernels [(3, 3), (3, 3), (5, 5), (5, 5), (5, 5)]; max_pool; lrelu; dense layers after flatten= 3

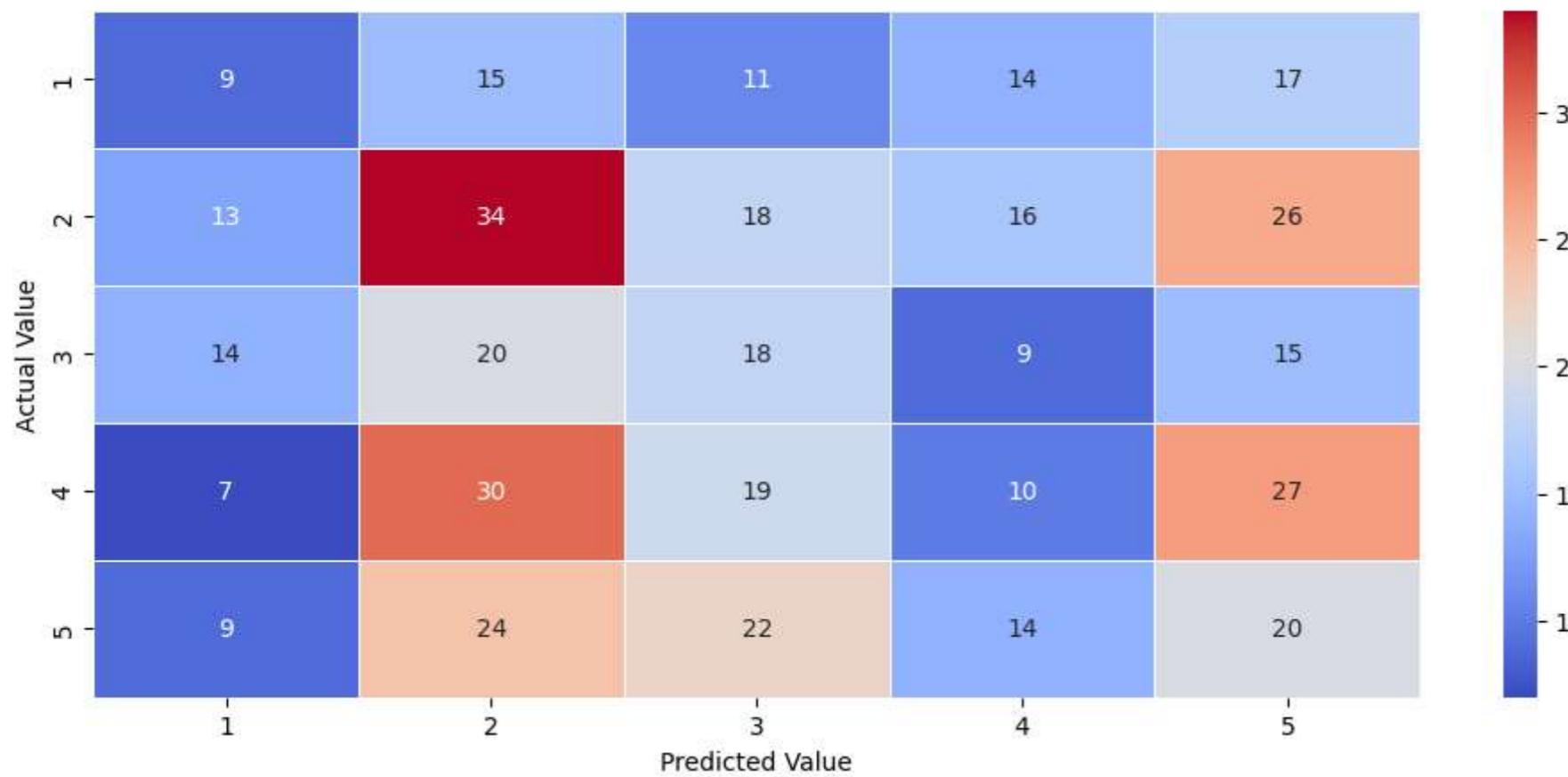
Loss vs epoch



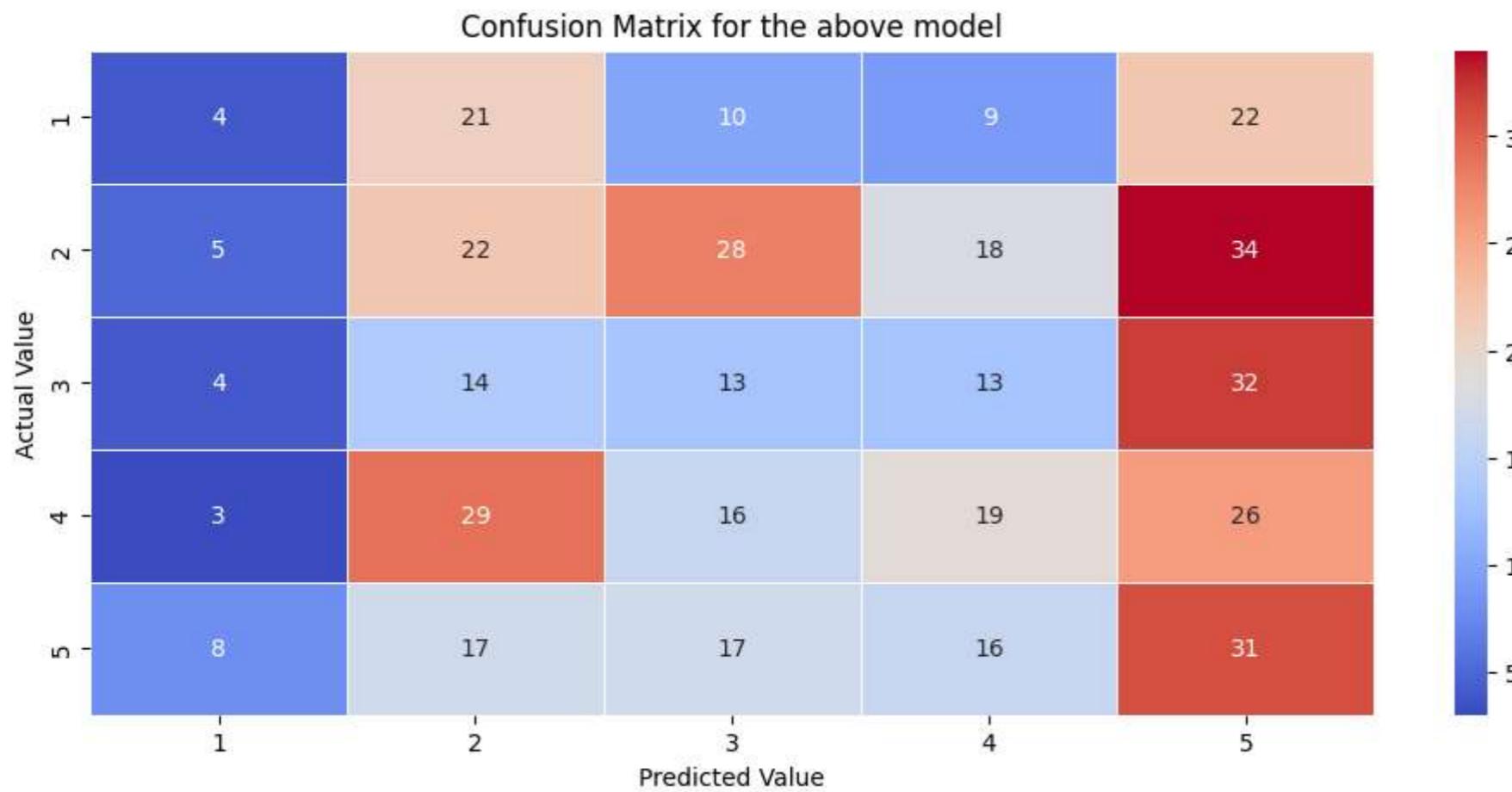
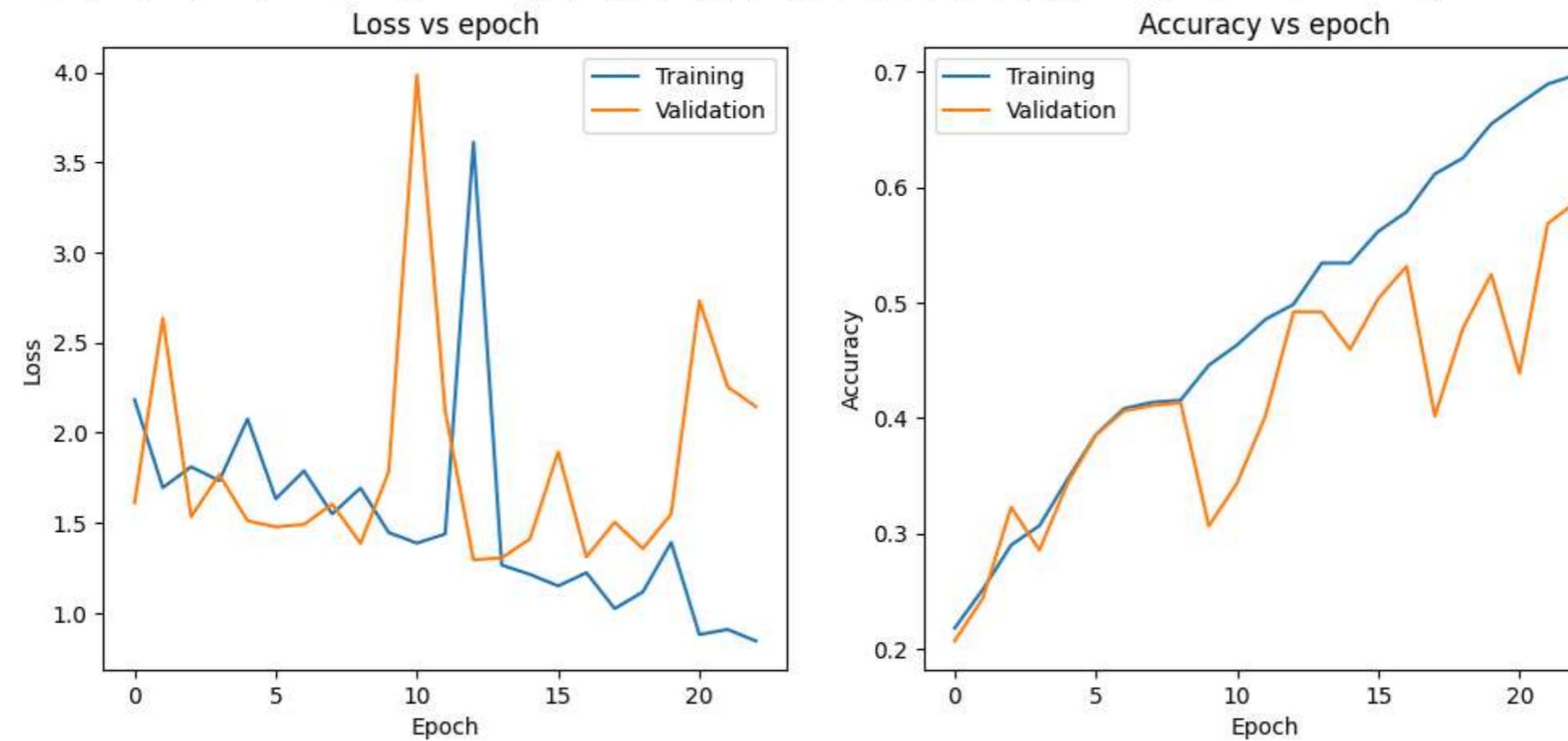
Accuracy vs epoch



Confusion Matrix for the above model



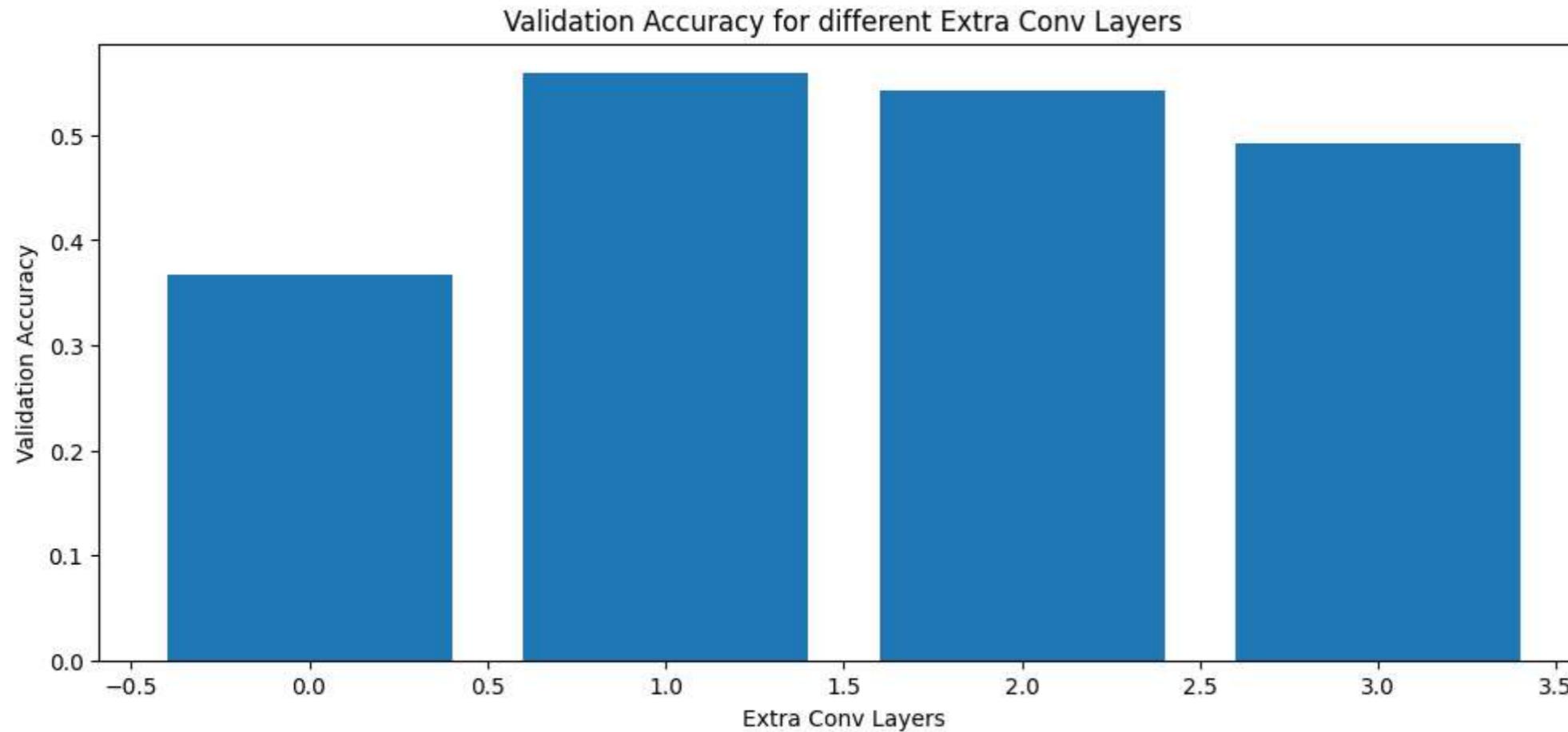
filters [16, 32, 64, 128, 256, 512]; kernels [(3, 3), (3, 3), (5, 5), (5, 5), (5, 5), (5, 5)]; max_pool; lrelu; dense layers after flatten= 3



```
In [ ]: result['Validation f1'] = \
    2*result['Validation Precision']*result['Validation Recall']/(result['Validation Precision'] + result['Validation Recall'])
result
```

Out[]:	Convolution kernel_size	Convolution filters size	Pooling Layers	Activation	FC layer (after Flatten)	Dropout Rate	Batch Normalization Present	Extra Conv Layers	Validation Loss	Validation Accuracy	Validation Precision	Validation Recall	Time Taken	Validation f1
0	[(3, 3), (3, 3), (5, 5)]	[16, 32, 64]	max	lrelu	3	0.1	True	0	1.553687	0.366589	0.500000	0.150812	56.404312	0.231729
1	[(3, 3), (3, 3), (5, 5), (5, 5)]	[16, 32, 64, 128]	max	lrelu	3	0.1	True	1	1.317321	0.559165	0.623616	0.392111	68.973295	0.481481
2	[(3, 3), (3, 3), (5, 5), (5, 5), (5, 5)]	[16, 32, 64, 128, 256]	max	lrelu	3	0.1	True	2	1.217815	0.542923	0.656566	0.452436	95.204106	0.535714
3	[(3, 3), (3, 3), (5, 5), (5, 5), (5, 5), (5, 5)]	[16, 32, 64, 128, 256, 512]	max	lrelu	3	0.1	True	3	1.294398	0.491879	0.629808	0.303944	119.813626	0.410016

```
In [ ]: plt.figure(figsize=(12, 5))
plt.bar(
    result['Extra Conv Layers'],
    result['Validation Accuracy'])
plt.ylabel('Validation Accuracy')
plt.xlabel('Extra Conv Layers')
plt.title("Validation Accuracy for different Extra Conv Layers")
plt.show()
```



```
In [ ]: best_extra_conv_layer = result.sort_values(
    by=['Validation Accuracy', 'Validation Loss'],
    ascending=[False, True]
)[['Extra Conv Layers']].iloc[0]

best_extra_conv_layer
```

Out[]: 1

Task 4.7

For the best set of parameters obtained here repeat the experimentation for color images. And visualize the test result.

```
In [ ]: result = pd.DataFrame(columns=[  
    'Convolution kernel_size',  
    'Convolution filters size',  
    'Pooling Layers',  
    'Activation',  
    'FC layer (after Flatten)',  
    'Dropout Rate',  
    'Batch Normalization Present',  
    'Extra Conv Layers',  
    'Color Mode',  
    'Validation Loss',  
    'Validation Accuracy',  
    'Validation Precision',  
    'Validation Recall',  
])  
  
convolution_filters = [16, 32, 64]  
  
val_loss, val_acc, val_precision, val_recall = train_model(  
    drop_rate=best_dropout,  
    conv_kernels=best_conv_kernel,  
    conv_filters=convolution_filters,  
    activation=best_activation,  
    pool=best_pool,  
    add_batch_norm=best_do_batch,  
    num_fc_layers=best_num_fc_layers,  
    extra_conv_layers=best_extra_conv_layer,  
    train_dataset=train_dataset,  
    val_dataset=val_dataset  
)  
  
result.loc[len(result.index)] = [  
    best_conv_kernel,  
    convolution_filters,  
    best_pool,  
    best_activation,  
    best_num_fc_layers,  
    best_dropout,  
    best_do_batch,  
    best_extra_conv_layer,  
    'grayscale',  
    val_loss,  
    val_acc,  
    val_precision,  
    val_recall  
]  
  
convolution_filters = [16, 32, 64]  
  
color_train_dataset = tf.keras.preprocessing.image_dataset_from_directory(  
    directory=DATASET_DIRECTORY_PATH,  
    labels="inferred",  
    label_mode="categorical", # output of model will be softmax categorical  
    color_mode="rgb",  
    batch_size=32,
```

```
    image_size=(80, 80),
    shuffle=True,
    seed=seed, # same seed for both dataset so that no overlap happens
    validation_split=0.1, # 90:10 split
    subset='training',
    crop_to_aspect_ratio=False,
)

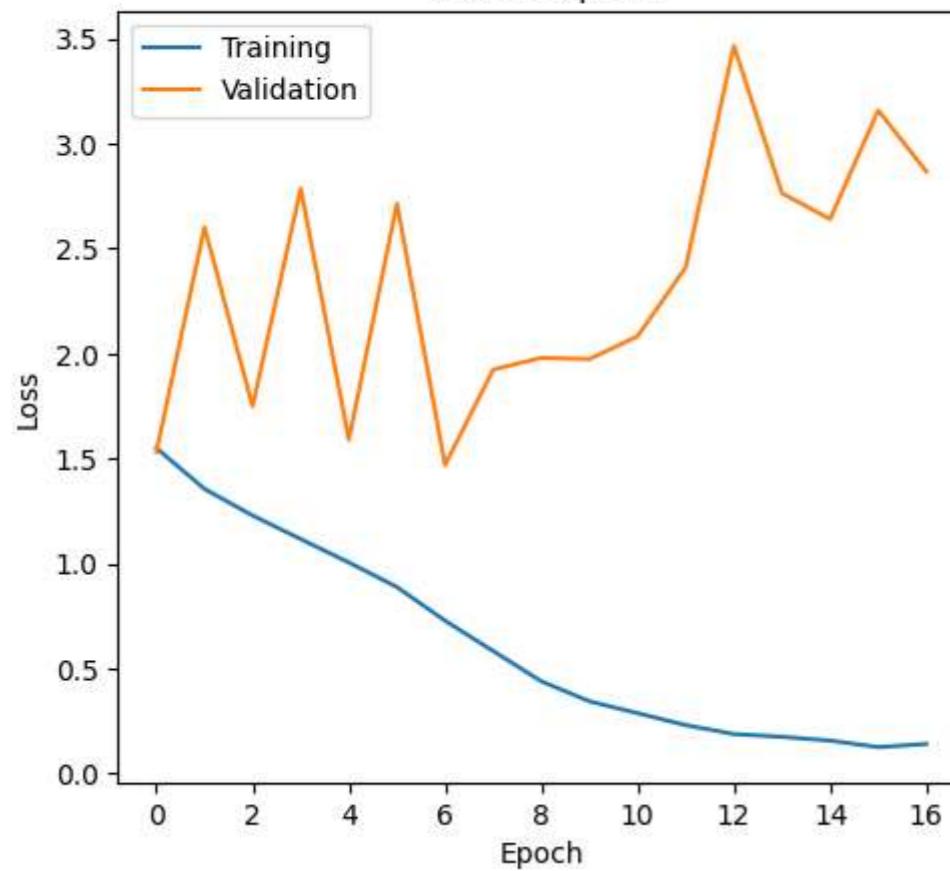
color_val_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    directory=DATASET_DIRECTORY_PATH,
    labels="inferred",
    label_mode="categorical", # output of model will be softmax categorical
    color_mode="rgb",
    batch_size=32,
    image_size=(80, 80),
    shuffle=True,
    seed=seed,
    validation_split=0.1, # 90:10 split
    subset='validation',
    crop_to_aspect_ratio=False,
)

val_loss, val_acc, val_precision, val_recall = train_model(
    drop_rate=best_dropout,
    conv_kernels=best_conv_kernel,
    conv_filters=convolution_filters,
    activation=best_activation,
    pool=best_pool,
    add_batch_norm=best_do_batch,
    num_fc_layers=best_num_fc_layers,
    extra_conv_layers=best_extra_conv_layer,
    train_dataset=train_dataset,
    val_dataset=val_dataset
)

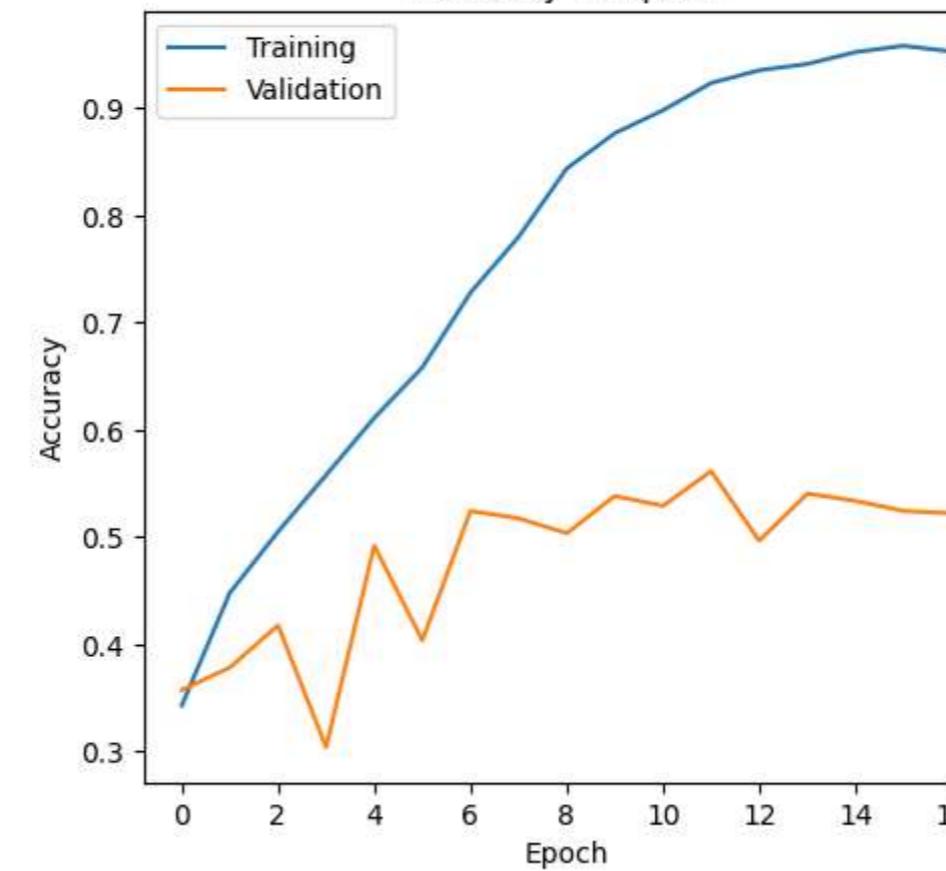
result.loc[len(result.index)] = [
    best_conv_kernel,
    convolution_filters,
    best_pool,
    best_activation,
    best_num_fc_layers,
    best_dropout,
    best_do_batch,
    best_extra_conv_layer,
    'rgb',
    val_loss,
    val_acc,
    val_precision,
    val_recall
]
```

filters [16, 32, 64, 128]; kernels [(3, 3), (3, 3), (5, 5), (5, 5)]; max_pool; lrelu; dense layers after flatten= 3

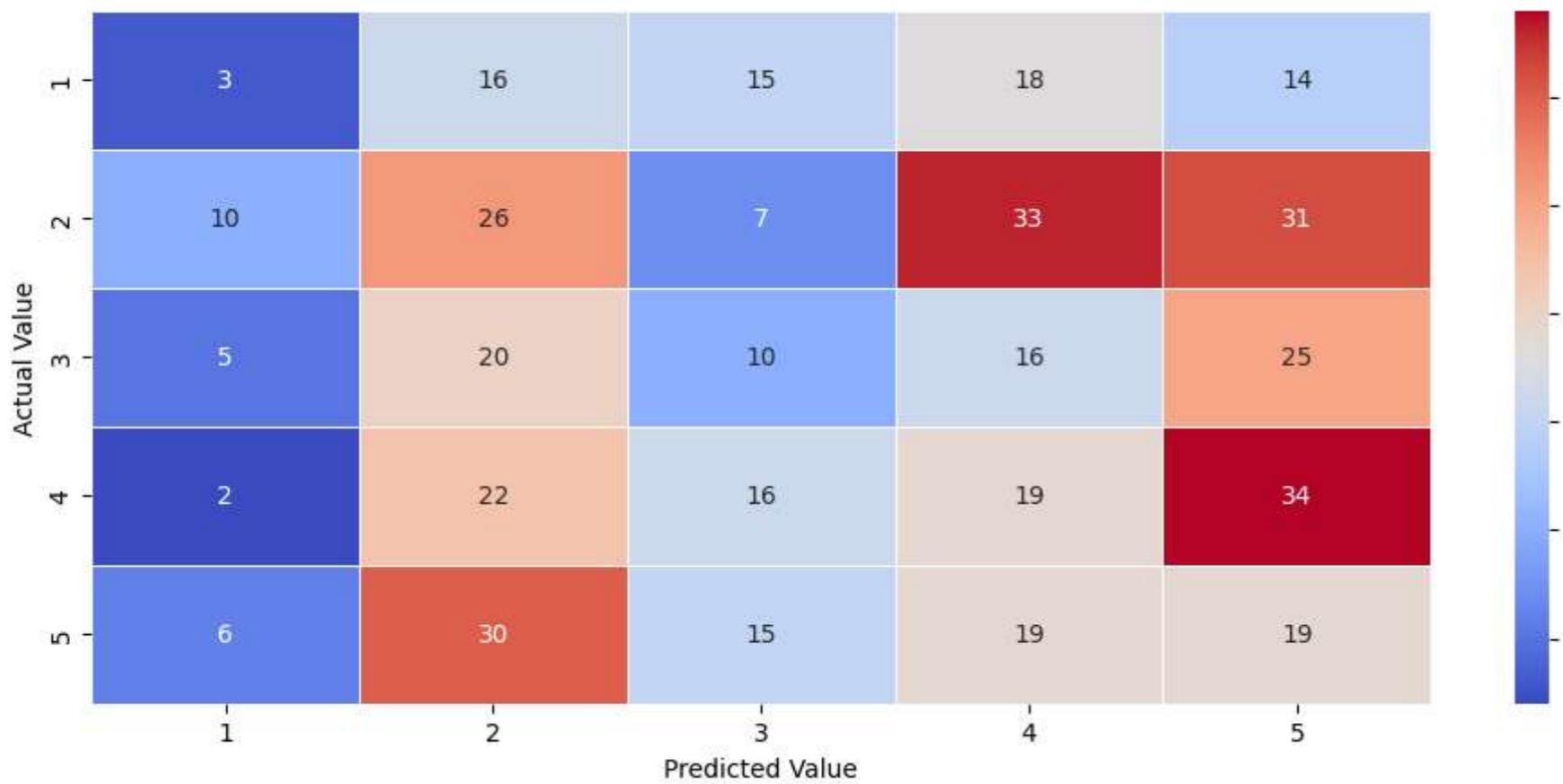
Loss vs epoch



Accuracy vs epoch



Confusion Matrix for the above model



Found 4317 files belonging to 5 classes.

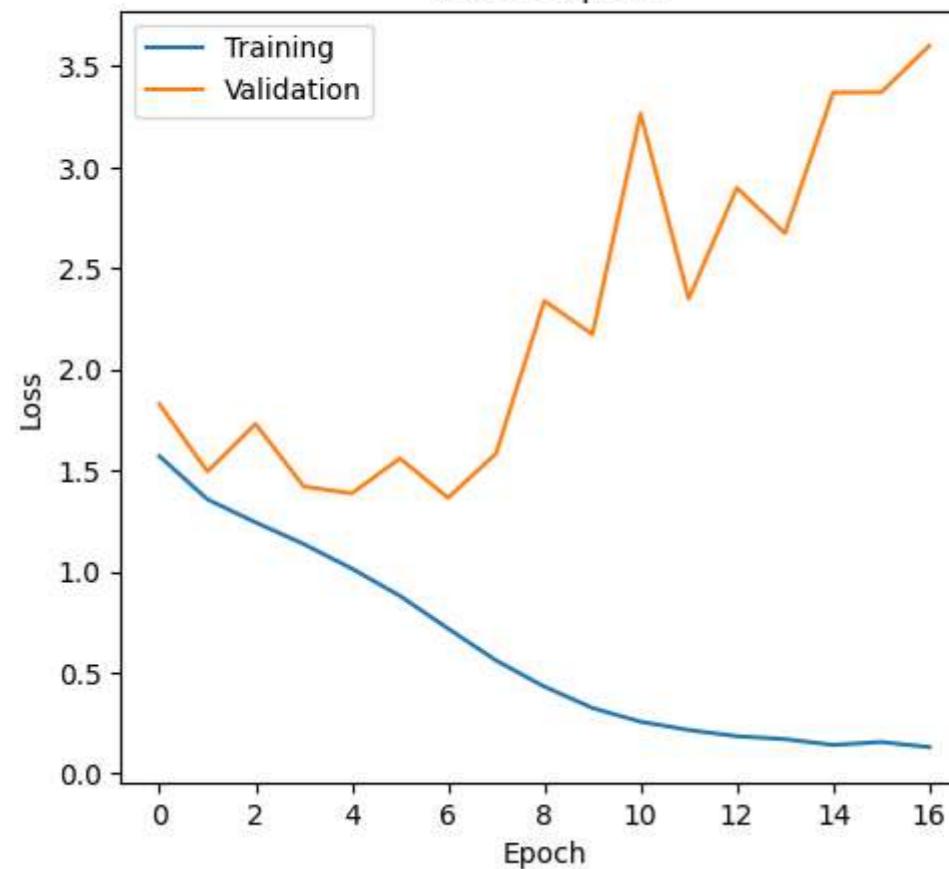
Using 3886 files for training.

Found 4317 files belonging to 5 classes.

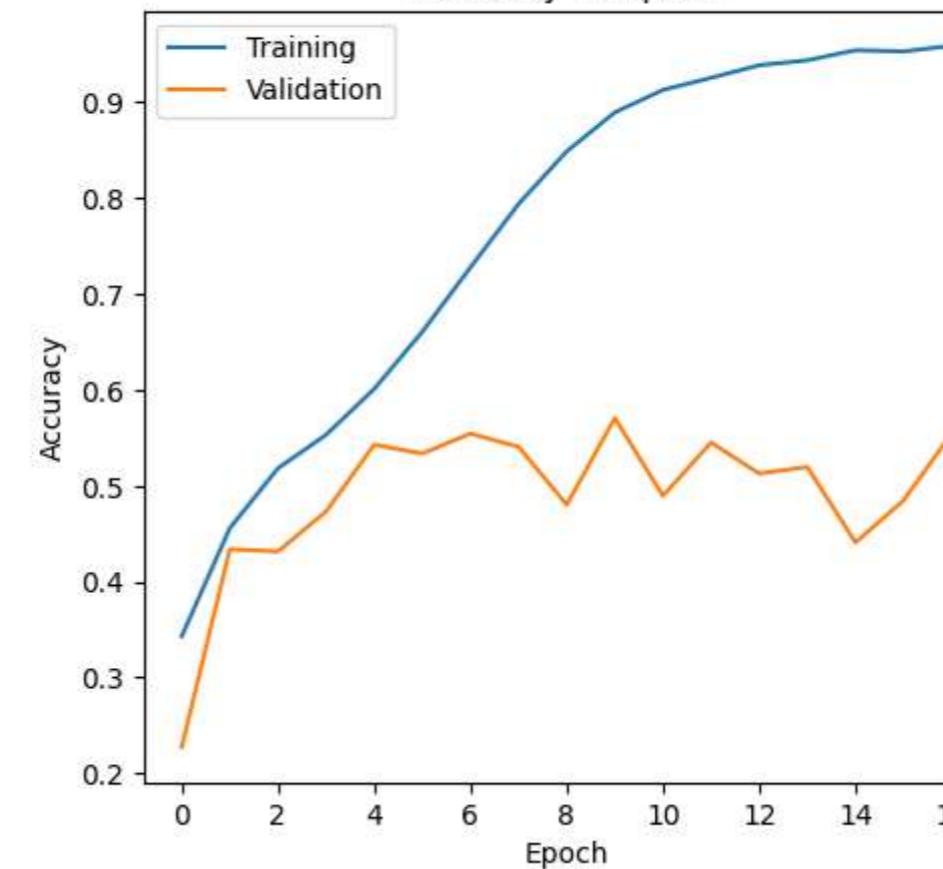
Using 431 files for validation.

filters [16, 32, 64, 128]; kernels [(3, 3), (3, 3), (5, 5), (5, 5), (5, 5)]; max_pool; lrelu; dense layers after flatten= 3

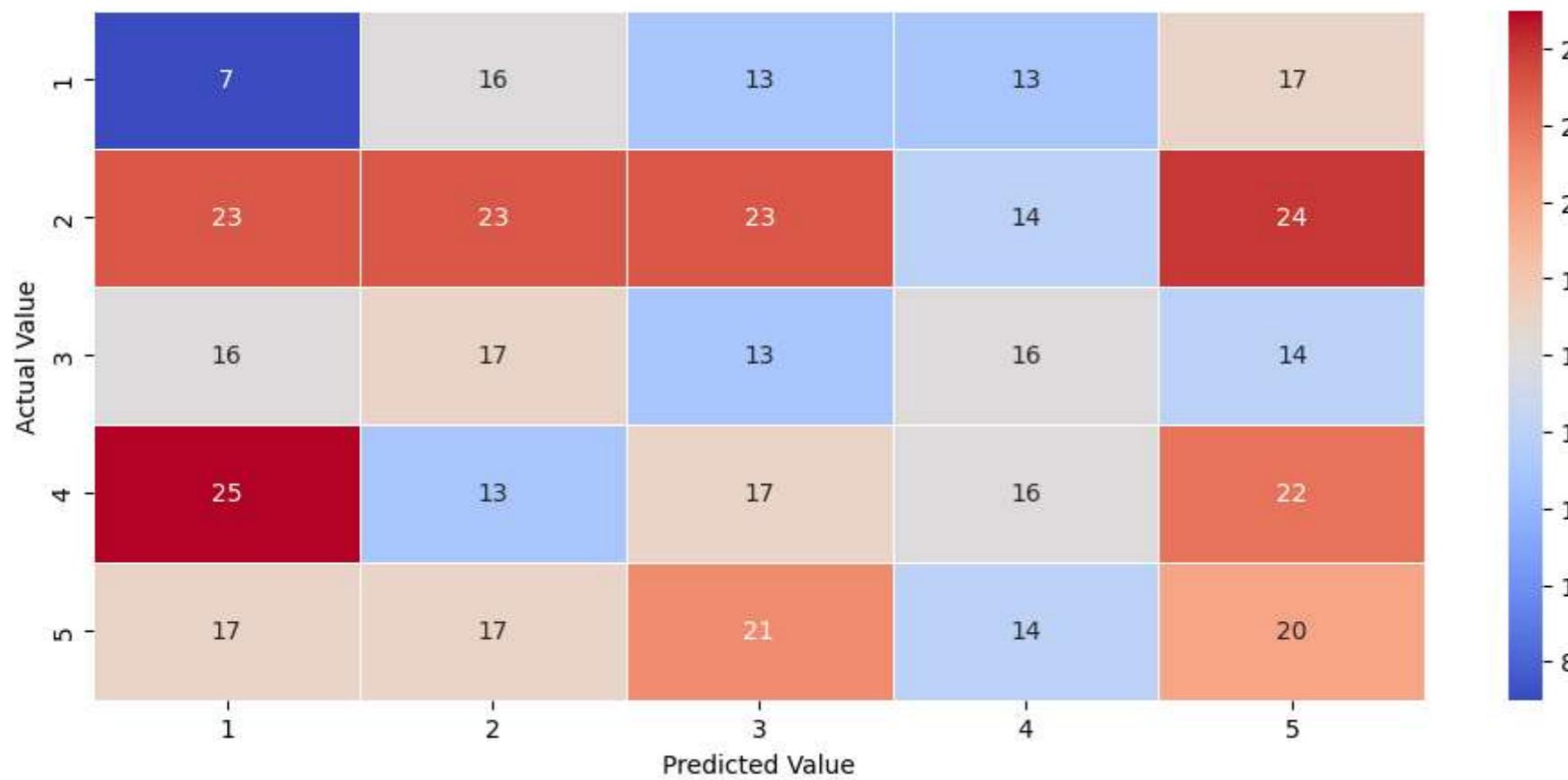
Loss vs epoch



Accuracy vs epoch



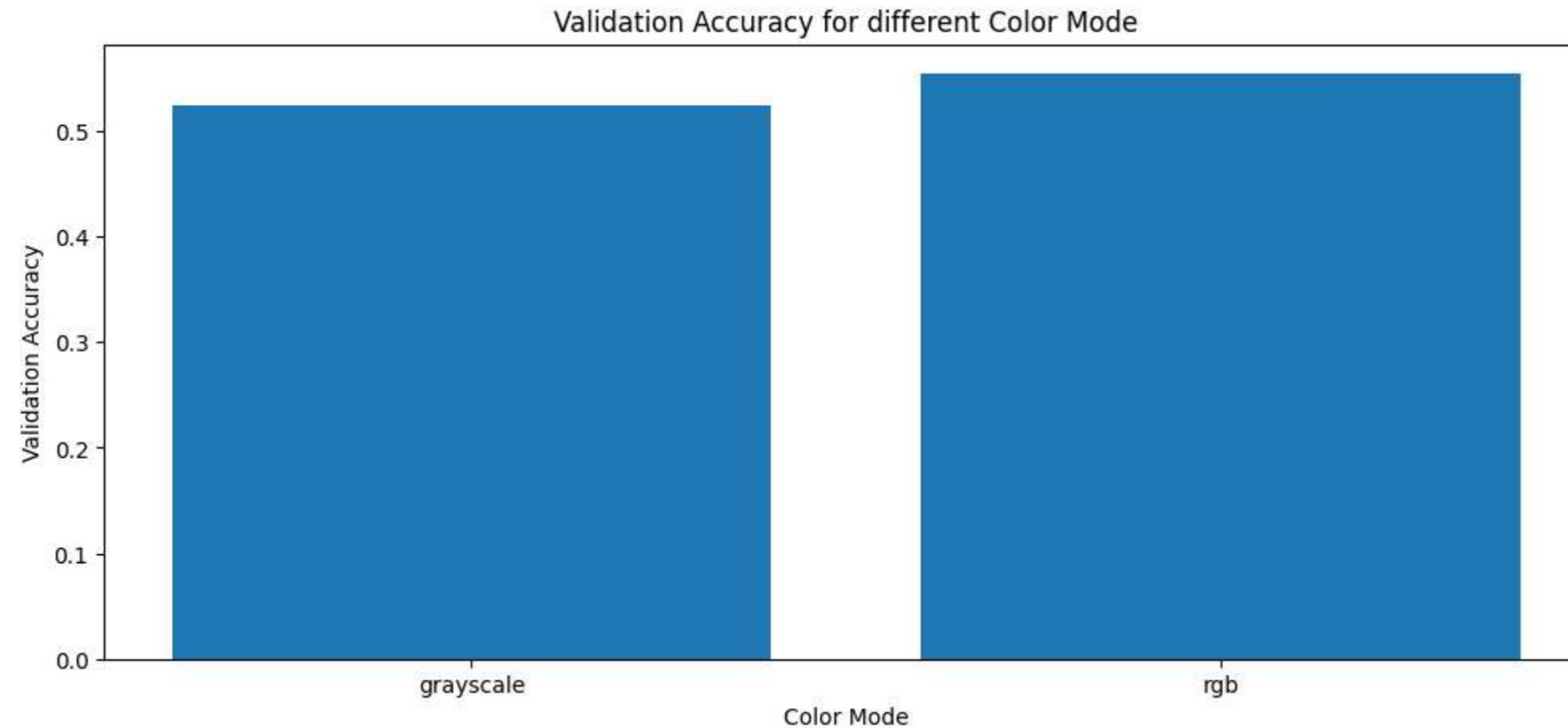
Confusion Matrix for the above model



```
In [ ]: result['Validation f1'] = \
    2*result['Validation Precision']*result['Validation Recall']/(result['Validation Precision'] + result['Validation Recall'])
result
```

Out[]:	Convolution kernel_size	Convolution filters size	Pooling Layers	Activation	FC layer (after Flatten)	Dropout Rate	Batch Normalization Present	Extra Conv Layers	Color Mode	Validation Loss	Validation Accuracy	Validation Precision	Validation Recall	Validation f1
0	[(3, 3), (3, 3), (5, 5), (5, 5), (5, 5)]	[16, 32, 64, 128]	max	lrelu	3	0.1	True	1	grayscale	1.468418	0.524362	0.597633	0.468677	0.525358
1	[(3, 3), (3, 3), (5, 5), (5, 5), (5, 5)]	[16, 32, 64, 128]	max	lrelu	3	0.1	True	1	rgb	1.364375	0.554524	0.590778	0.475638	0.526992

```
In [ ]: plt.figure(figsize=(12, 5))
plt.bar(
    result['Color Mode'],
    result['Validation Accuracy'])
plt.ylabel('Validation Accuracy')
plt.xlabel('Color Mode')
plt.title("Validation Accuracy for different Color Mode")
plt.show()
```



Task 5

Plot the graph for the loss vs epoch and accuracy(train, test set) vs epoch for all the above cases. Also, plot the accuracy for all experimentation in a bar graph along with the confusion matrix and F1 score

Task 6

For the best model on the MINST dataset in Assignment 5, train a model with MNIST data using the best set of parameters obtained in Question 4. Compare the test accuracy and also the self-created images.

```
In [ ]: import numpy as np
import tensorflow as tf
```

```

def create_dataset(X, Y, batch_size=32):
    """ Create train and test TF dataset from X and Y
        The prefetch overlays the preprocessing and model execution of a training step.
        While the model is executing training step s, the input pipeline is reading the data for step s+1.
        AUTOTUNE automatically tune the number for sample which are prefetched automatically.

    Keyword arguments:
    X -- numpy array
    Y -- numpy array
    batch_size -- integer
    """
    AUTOTUNE = tf.data.experimental.AUTOTUNE

    dataset = tf.data.Dataset.from_tensor_slices((X, Y))
    dataset = dataset.shuffle(buffer_size=1000, reshuffle_each_iteration=True)
    dataset = dataset.batch(batch_size).prefetch(AUTOTUNE)

    return dataset

```

```

In [ ]: import pandas as pd
from tensorflow.image import resize

mnist_data = tf.keras.datasets.mnist.load_data()
(x_train, y_train), (x_test, y_test) = mnist_data

#taking Less data due to memory limitation
x_train = x_train[0:1000]
x_test = x_test[0:2000]
y_train = y_train[0:1000]
y_test = y_test[0:2000]

#resizing to 80x80 and mapping 0-255 to 0-1
x_train = np.reshape(x_train, (-1, 28, 28, 1))
x_train = x_train/255
x_train = resize(x_train, (80,80))

x_test = np.reshape(x_test, (-1, 28, 28, 1))
x_test = x_test/255
x_test = resize(x_test, (80,80))

y_train = pd.get_dummies(y_train).to_numpy()
y_test = pd.get_dummies(y_test).to_numpy()

mnist_train_dataset=create_dataset(x_train, y_train)
mnist_val_dataset=create_dataset(x_test, y_test)

```

```

In [ ]: convolution_filters = [16, 32, 64]
num_class = 10

val_loss, val_acc, val_precision, val_recall, model = train_model(
    drop_rate=best_dropout,
    conv_kernels=best_conv_kernel,
    conv_filters=convolution_filters,
    activation=best_activation,
    pool=best_pool,
    add_batch_norm=best_do_batch,
    num_fc_layers=best_num_fc_layers,
    extra_conv_layers=best_extra_conv_layer,

```

```

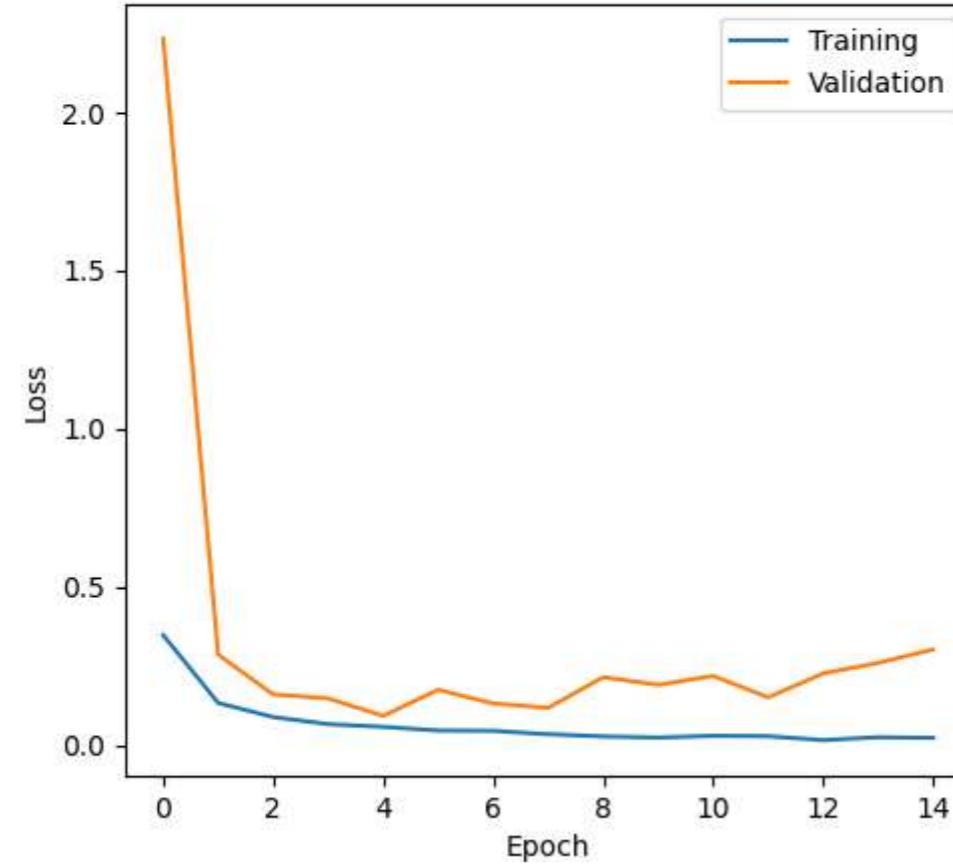
        train_dataset=mnist_train_dataset,
        val_dataset=mnist_val_dataset,
        give_model=True
    )

    print(f"val_loss= {val_loss}\nval_acc={val_acc}\nval_precision={val_precision}\nval_recall={val_recall}")

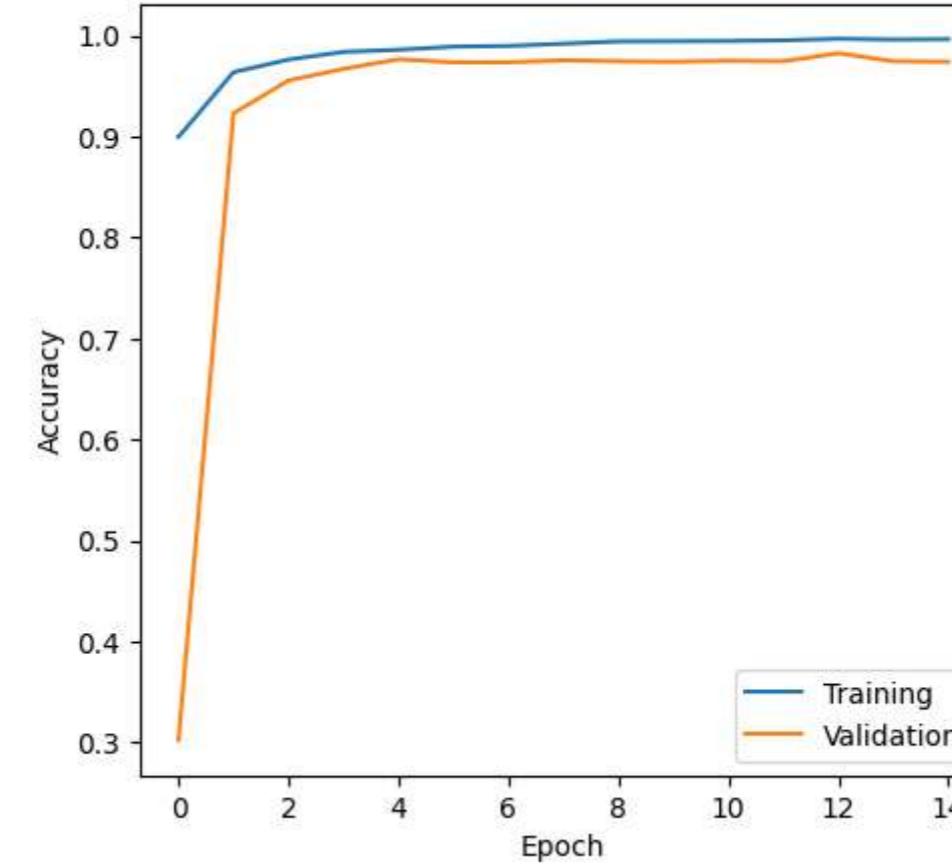
```

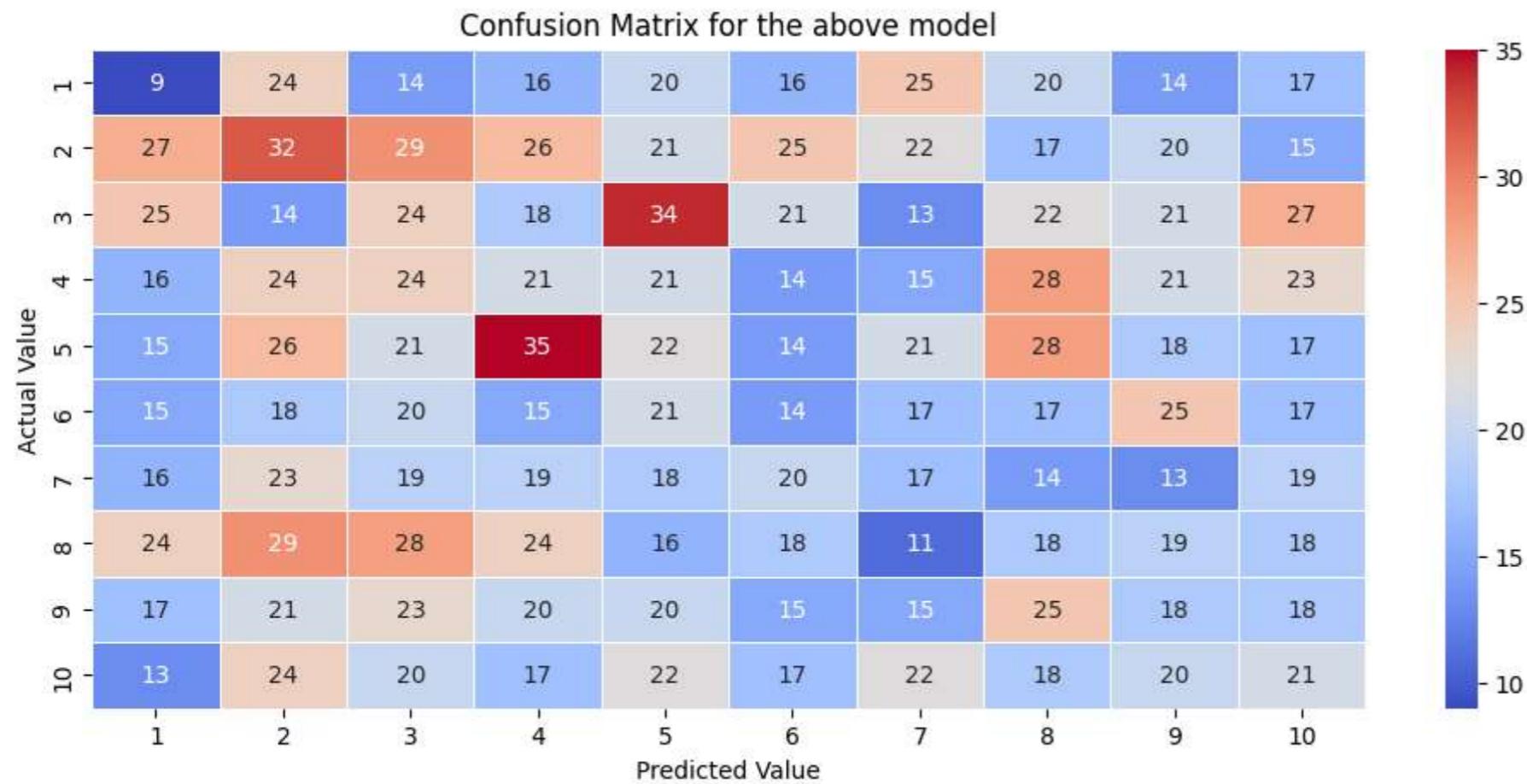
filters [16, 32, 64, 128]; kernels [(3, 3), (3, 3), (5, 5), (5, 5), (5, 5), (5, 5), (5, 5)]; max_pool; lrelu; dense layers after flatten= 3

Loss vs epoch



Accuracy vs epoch





```
val_loss= 0.09174732118844986
val_acc=0.9764999747276306
val_precision=0.977477490901947
val_recall=0.9764999747276306
```

In []:

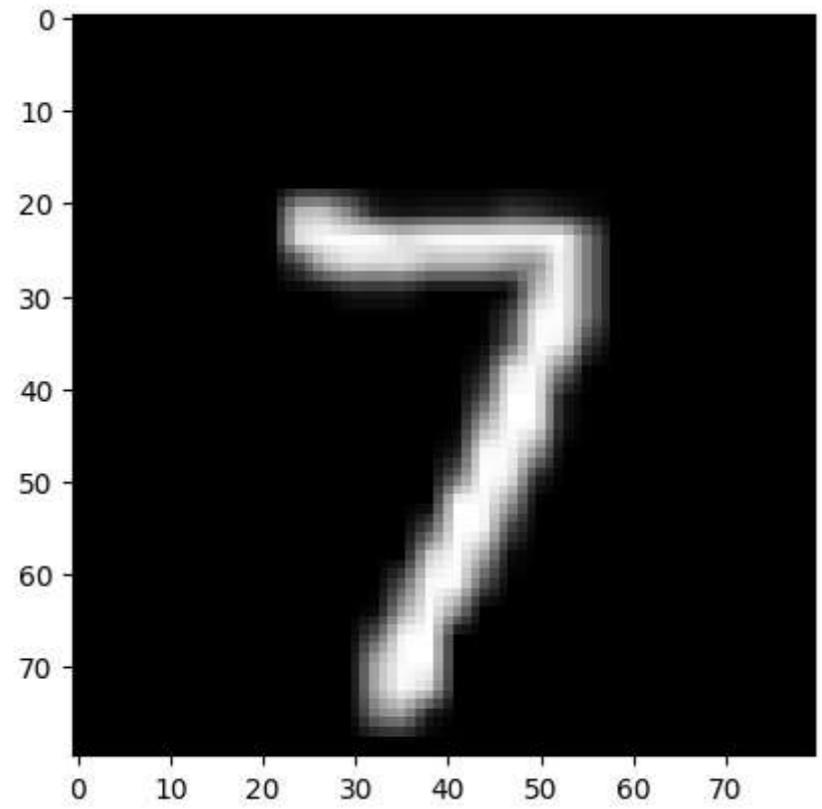
```
import random

random_idx = random.sample(range(0, len(x_test)), 10)

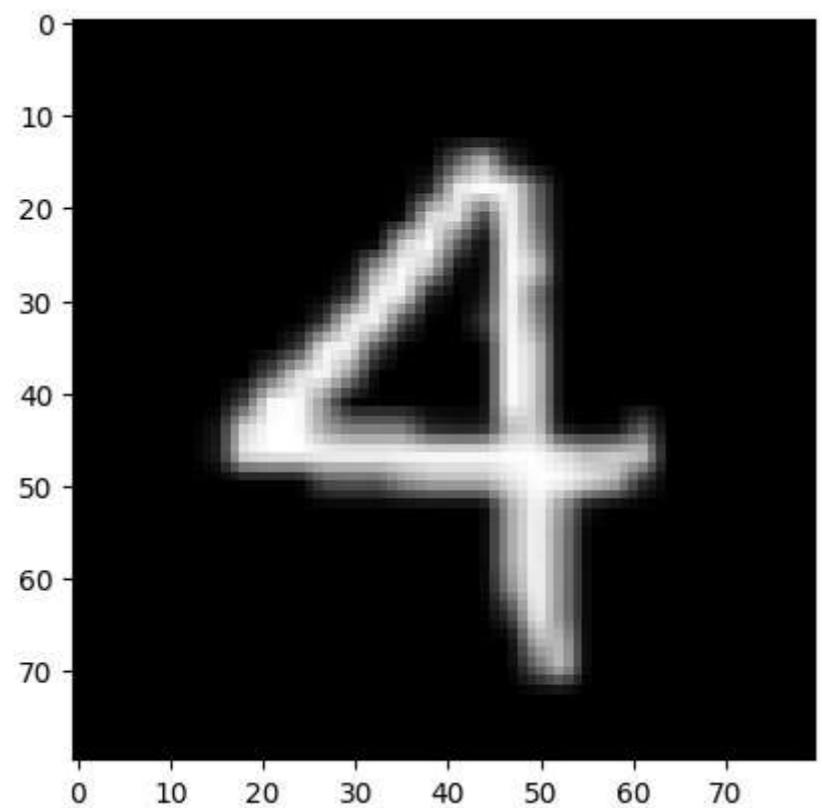
img_to_predict = np.array([x_test[idx] for idx in random_idx])

categorical_predictions = model.predict(img_to_predict)

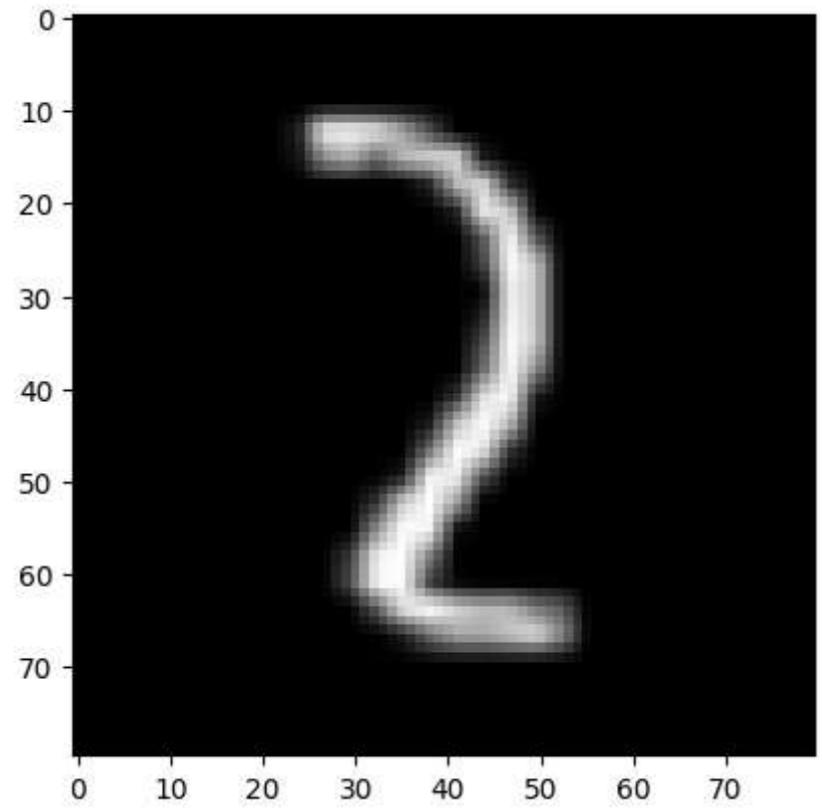
for img, cat_pred in zip(img_to_predict, categorical_predictions):
    plt.imshow(img, cmap='gray')
    plt.show()
    pred = np.argmax(cat_pred)
    print(f"predict = {pred}")
```



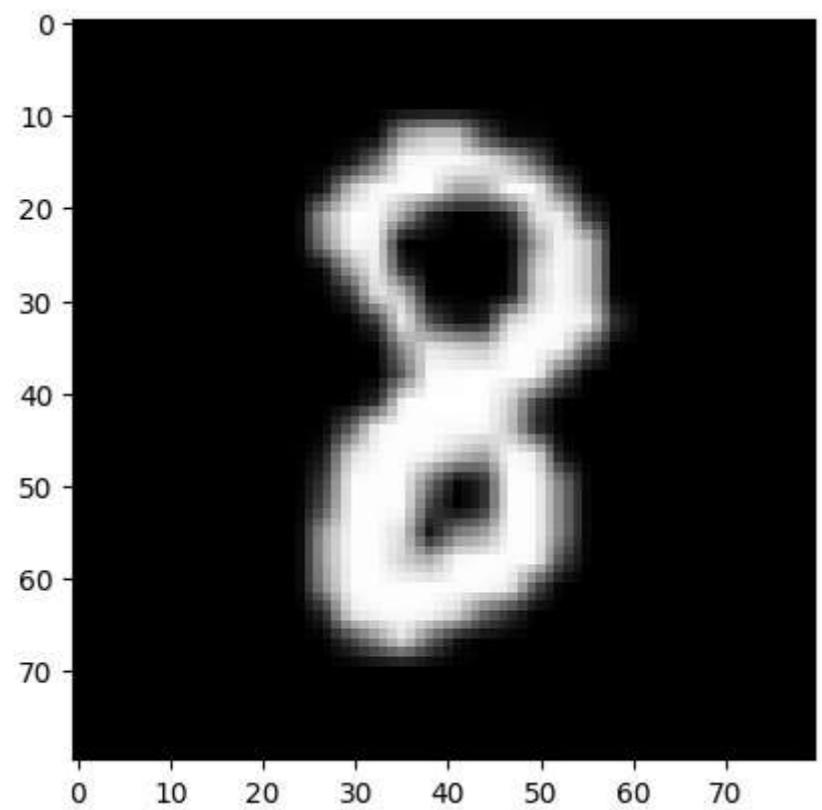
predict = 7



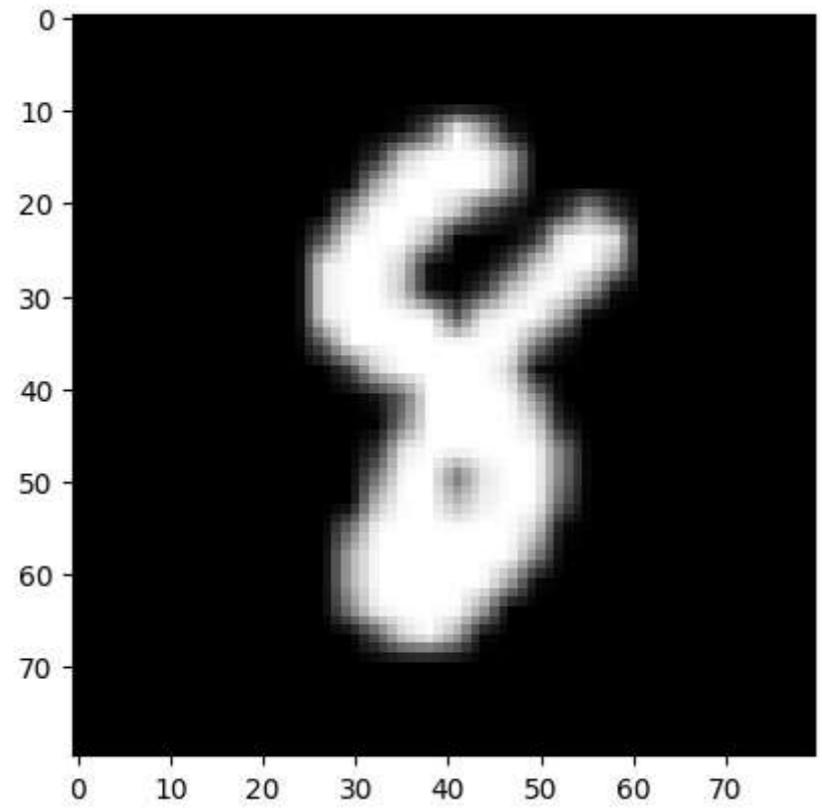
predict = 4



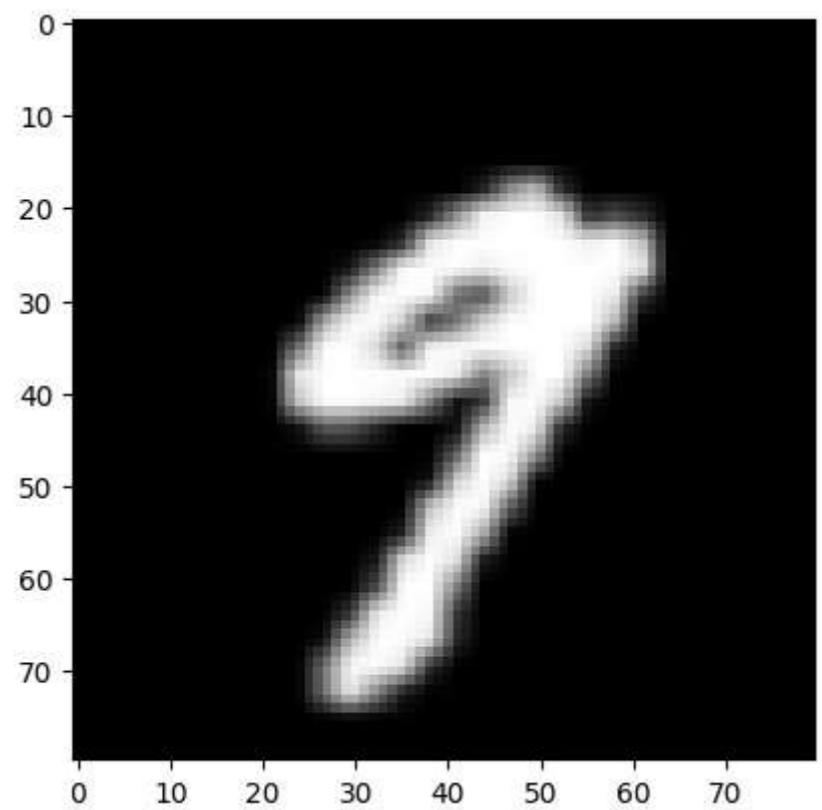
predict = 2



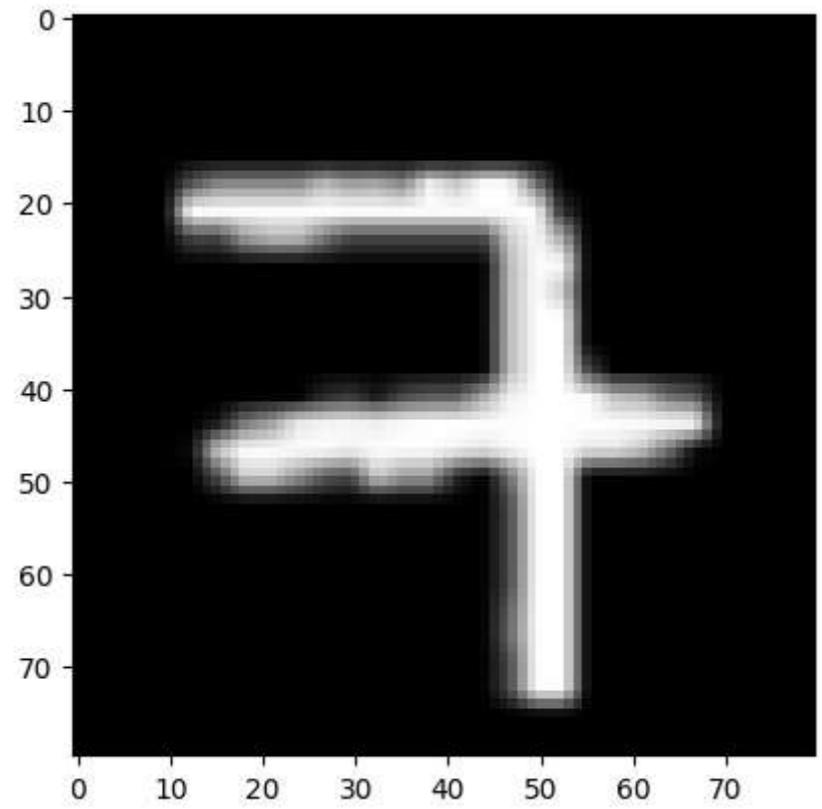
predict = 8



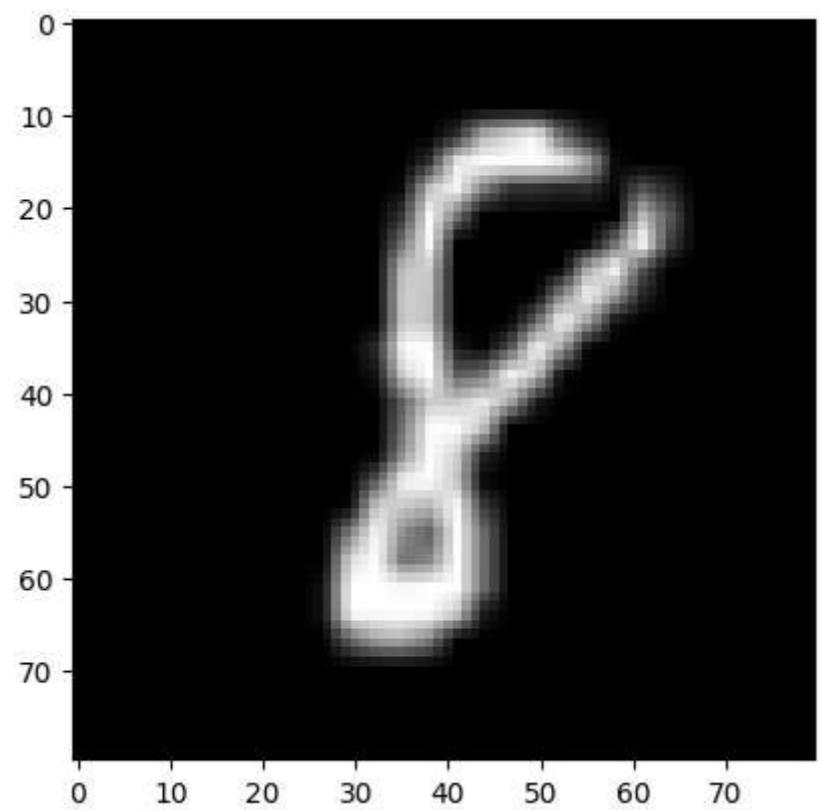
predict = 8



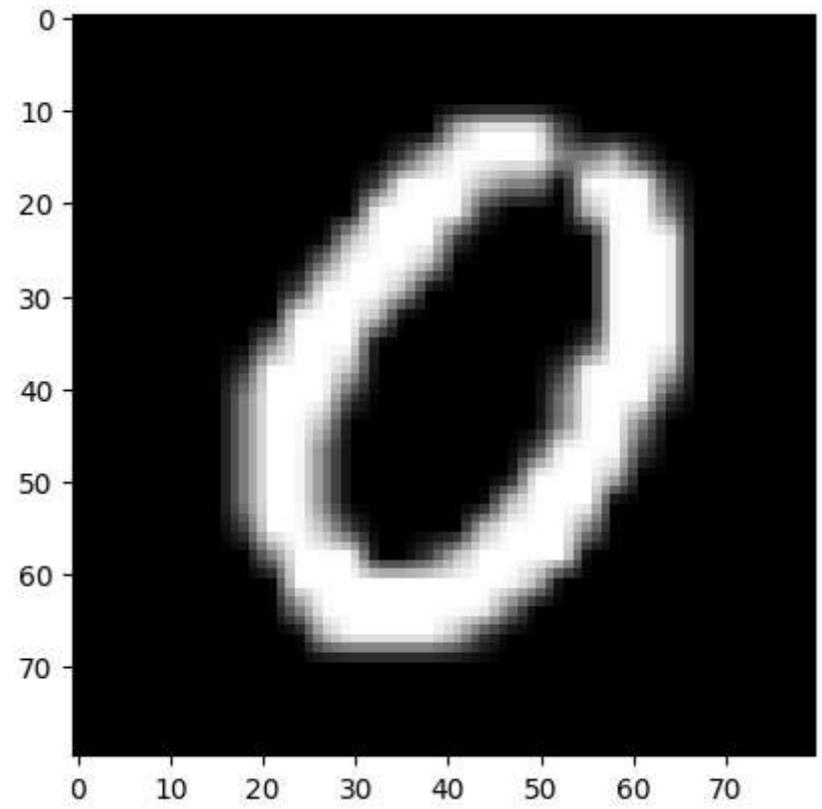
predict = 9



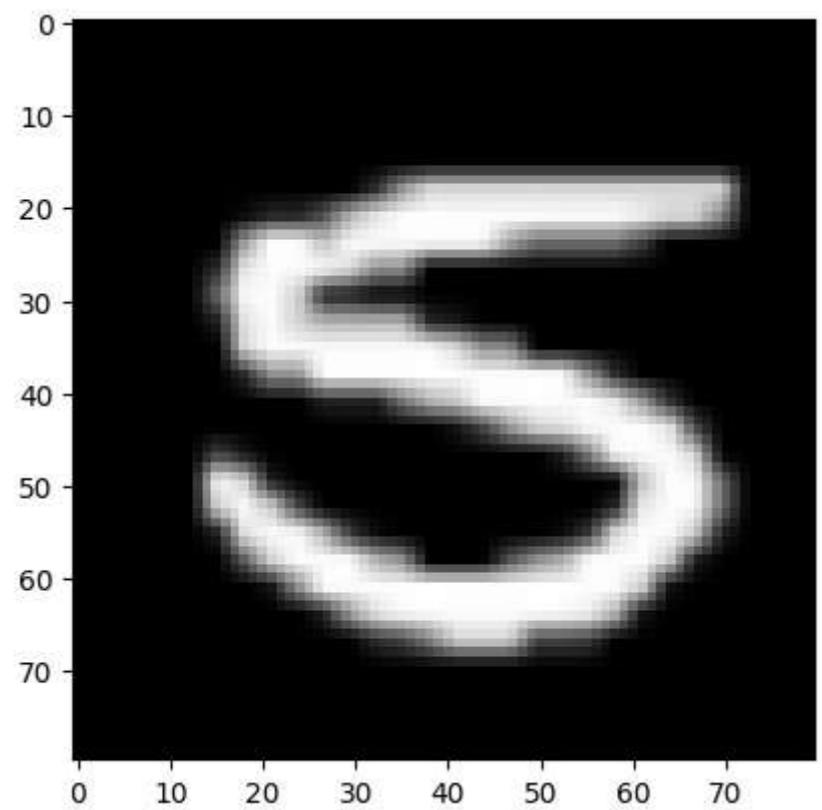
predict = 7



predict = 8



predict = 0



predict = 5