```
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, Activation
from keras.utils import to_categorical
from keras.callbacks import ModelCheckpoint
from keras.optimizers import Adam
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
```

```
# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 18s 0us/step
```

```
# Normalize the input data by scaling pixel values to between 0 and 1
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

+ Code + Text

```
# Convert the label vectors to binary class matrices
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

```
# Split the training data to create a validation set (20% of training data)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=42)
```

```
# Model 1: Basic CNN Model
model1 = Sequential([
    # First convolutional block
    Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(32, 32, 3)),
    Conv2D(32, (3, 3), padding='same', activation='relu'),
    MaxPooling2D((2, 2)),

    # Second convolutional block
    Conv2D(64, (3, 3), padding='same', activation='relu'),
    Conv2D(64, (3, 3), padding='same', activation='relu'),
    MaxPooling2D((2, 2)),

    # Flattening layer
    Flatten(),

    # Fully connected layer
    Dense(512, activation='relu'),
    Dense(10, activation='softmax')
])
```

```
# Compile the model
model1.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Set up model checkpoint to save the best model
checkpoint1 = ModelCheckpoint('best_model1.h5', monitor='val_loss', save_best_only=True, mode='min')
```

```
# Train the model
history1 = model1.fit(x_train, y_train, epochs=50, batch_size=32, validation_data=(x_val, y_val), callbacks=[checkpoint1])
```

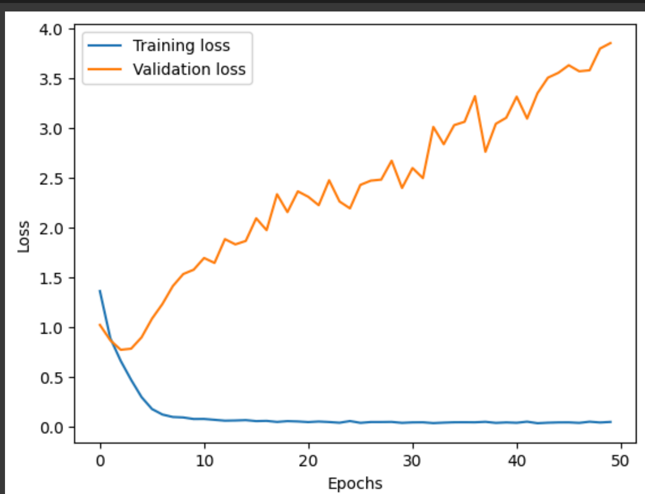
```
Epoch 1/50
1250/1250 [==============================] - 13s 6ms/step - loss: 1.3616 - accuracy: 0.5049 - val_loss: 1.0220 - val_accuracy: 0.6421
Epoch 2/50
  23/1250 [..............................] - ETA: 5s - loss: 0.9832 - accuracy: 0.6576/usr/local/lib/python3.10/dist-packages/keras/src/engine/trair
  saving_api.save_model(
1250/1250 [==============================] - 8s 6ms/step - loss: 0.8883 - accuracy: 0.6894 - val_loss: 0.8681 - val_accuracy: 0.6981
Epoch 3/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6610 - accuracy: 0.7674 - val_loss: 0.7727 - val_accuracy: 0.7346
Epoch 4/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.4703 - accuracy: 0.8360 - val_loss: 0.7837 - val_accuracy: 0.7426
Epoch 5/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.2975 - accuracy: 0.8963 - val_loss: 0.8980 - val_accuracy: 0.7529
Epoch 6/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.1782 - accuracy: 0.9382 - val_loss: 1.0848 - val_accuracy: 0.7474
Epoch 7/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.1230 - accuracy: 0.9583 - val_loss: 1.2322 - val_accuracy: 0.7364
Epoch 8/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.0984 - accuracy: 0.9673 - val_loss: 1.4119 - val_accuracy: 0.7299
Epoch 9/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.0937 - accuracy: 0.9682 - val_loss: 1.5327 - val_accuracy: 0.7323
Epoch 10/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.0782 - accuracy: 0.9737 - val_loss: 1.5762 - val_accuracy: 0.7368
Epoch 11/50
1250/1250 [==============================] - 8s 6ms/step - loss: 0.0788 - accuracy: 0.9744 - val_loss: 1.6937 - val_accuracy: 0.7271
Epoch 12/50
1250/1250 [==============================] - 7s 5ms/step - loss: 0.0699 - accuracy: 0.9771 - val_loss: 1.6440 - val_accuracy: 0.7304
Epoch 13/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.0613 - accuracy: 0.9800 - val_loss: 1.8829 - val_accuracy: 0.7304
Epoch 14/50
1250/1250 [==============================] - 7s 5ms/step - loss: 0.0633 - accuracy: 0.9793 - val_loss: 1.8301 - val_accuracy: 0.7224
```

```
Epoch 15/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.0669 - accuracy: 0.9786 - val_loss: 1.8648 - val_accuracy: 0.7315
Epoch 16/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.0570 - accuracy: 0.9815 - val_loss: 2.0917 - val_accuracy: 0.7270
Epoch 17/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.0594 - accuracy: 0.9806 - val_loss: 1.9726 - val_accuracy: 0.7386
Epoch 18/50
1250/1250 [==============================] - 8s 6ms/step - loss: 0.0488 - accuracy: 0.9846 - val_loss: 2.3333 - val_accuracy: 0.7266
Epoch 19/50
1250/1250 [==============================] - 7s 5ms/step - loss: 0.0569 - accuracy: 0.9825 - val_loss: 2.1539 - val_accuracy: 0.7129
Epoch 20/50
1250/1250 [==============================] - 8s 6ms/step - loss: 0.0536 - accuracy: 0.9834 - val_loss: 2.3623 - val_accuracy: 0.7116
Epoch 21/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.0472 - accuracy: 0.9851 - val_loss: 2.3075 - val_accuracy: 0.7239
Epoch 22/50
1250/1250 [==============================] - 8s 6ms/step - loss: 0.0530 - accuracy: 0.9838 - val_loss: 2.2230 - val_accuracy: 0.7323
Epoch 23/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.0478 - accuracy: 0.9855 - val_loss: 2.4735 - val_accuracy: 0.7297
Epoch 24/50
1250/1250 [==============================] - 7s 5ms/step - loss: 0.0408 - accuracy: 0.9883 - val_loss: 2.2602 - val_accuracy: 0.7344
Epoch 25/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.0576 - accuracy: 0.9838 - val_loss: 2.1913 - val_accuracy: 0.7251
Epoch 26/50
1250/1250 [==============================] - 7s 5ms/step - loss: 0.0397 - accuracy: 0.9882 - val_loss: 2.4270 - val_accuracy: 0.7283
Epoch 27/50
1250/1250 [==============================] - 7s 6ms/step - loss: 0.0477 - accuracy: 0.9857 - val_loss: 2.4696 - val_accuracy: 0.7250
Epoch 28/50
1250/1250 [==============================] - 6s 5ms/step - loss: 0.0478 - accuracy: 0.9851 - val_loss: 2.4799 - val_accuracy: 0.7209
```

```
# Evaluate the model on the test set
test_loss1, test_acc1 = model1.evaluate(x_test, y_test)
print(f"Model 1 - Test accuracy: {test_acc1}")
print(f"Model 1 - Test loss: {test_loss1}")
```

```
313/313 [==============================] - 1s 3ms/step - loss: 3.9360 - accuracy: 0.7162
Model 1 - Test accuracy: 0.7161999940872192
Model 1 - Test loss: 3.935969114303589
```

```
# Plot training and validation loss for all epochs
plt.plot(history1.history['loss'], label='Training loss')
plt.plot(history1.history['val_loss'], label='Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
# Model 2: CNN with Data Augmentation
model2 = Sequential([
    # First convolutional block
    Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(32, 32, 3)),
    Conv2D(32, (3, 3), padding='same', activation='relu'),
    MaxPooling2D((2, 2)),

    # Second convolutional block
    Conv2D(64, (3, 3), padding='same', activation='relu'),
    Conv2D(64, (3, 3), padding='same', activation='relu'),
    MaxPooling2D((2, 2)),

    # Flattening layer
    Flatten(),

    # Fully connected layer
    Dense(512, activation='relu'),
    Dense(10, activation='softmax')
])
```

```
# Compile the model
model2.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Set up model checkpoint to save the best model
checkpoint2 = ModelCheckpoint('best_model2.h5', monitor='val_loss', save_best_only=True, mode='min')
```

```
# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)
datagen.fit(x_train)
```

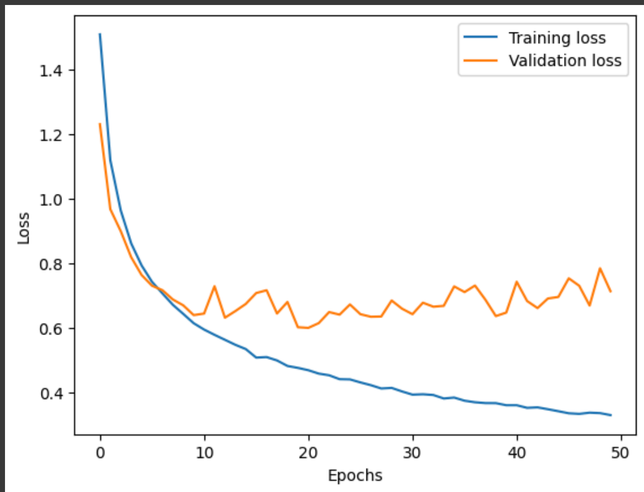
```
# Train the model with data augmentation
history2 = model2.fit(datagen.flow(x_train, y_train, batch_size=32),
                      epochs=50,
                      validation_data=(x_val, y_val),
                      callbacks=[checkpoint2])
```

```
Epoch 1/50
1250/1250 [==============================] - 35s 27ms/step - loss: 1.5089 - accuracy: 0.4503 - val_loss: 1.2309 - val_accuracy: 0.5587
Epoch 2/50
1250/1250 [==============================] - 32s 26ms/step - loss: 1.1191 - accuracy: 0.5986 - val_loss: 0.9676 - val_accuracy: 0.6535
Epoch 3/50
1250/1250 [==============================] - 31s 25ms/step - loss: 0.9639 - accuracy: 0.6590 - val_loss: 0.8998 - val_accuracy: 0.6928
Epoch 4/50
1250/1250 [==============================] - 32s 25ms/step - loss: 0.8620 - accuracy: 0.6931 - val_loss: 0.8193 - val_accuracy: 0.7220
Epoch 5/50
1250/1250 [==============================] - 32s 26ms/step - loss: 0.7937 - accuracy: 0.7184 - val_loss: 0.7642 - val_accuracy: 0.7345
Epoch 6/50
1250/1250 [==============================] - 32s 25ms/step - loss: 0.7429 - accuracy: 0.7396 - val_loss: 0.7309 - val_accuracy: 0.7486
Epoch 7/50
1250/1250 [==============================] - 32s 25ms/step - loss: 0.7072 - accuracy: 0.7527 - val_loss: 0.7170 - val_accuracy: 0.7583
Epoch 8/50
1250/1250 [==============================] - 33s 26ms/step - loss: 0.6718 - accuracy: 0.7649 - val_loss: 0.6883 - val_accuracy: 0.7673
Epoch 9/50
1250/1250 [==============================] - 32s 25ms/step - loss: 0.6437 - accuracy: 0.7726 - val_loss: 0.6701 - val_accuracy: 0.7786
Epoch 10/50
1250/1250 [==============================] - 32s 26ms/step - loss: 0.6148 - accuracy: 0.7847 - val_loss: 0.6399 - val_accuracy: 0.7864
Epoch 11/50
1250/1250 [==============================] - 31s 25ms/step - loss: 0.5952 - accuracy: 0.7921 - val_loss: 0.6447 - val_accuracy: 0.7874
Epoch 12/50
1250/1250 [==============================] - 31s 25ms/step - loss: 0.5789 - accuracy: 0.7975 - val_loss: 0.7293 - val_accuracy: 0.7730
Epoch 13/50
1250/1250 [==============================] - 32s 26ms/step - loss: 0.5635 - accuracy: 0.8020 - val_loss: 0.6319 - val_accuracy: 0.7987
Epoch 14/50
1250/1250 [==============================] - 31s 25ms/step - loss: 0.5480 - accuracy: 0.8076 - val_loss: 0.6524 - val_accuracy: 0.7868
Epoch 15/50
1250/1250 [==============================] - 31s 25ms/step - loss: 0.5348 - accuracy: 0.8091 - val_loss: 0.6745 - val_accuracy: 0.7902
Epoch 16/50
1250/1250 [==============================] - 32s 26ms/step - loss: 0.5084 - accuracy: 0.8221 - val_loss: 0.7082 - val_accuracy: 0.7735
Epoch 17/50
1250/1250 [==============================] - 30s 24ms/step - loss: 0.5100 - accuracy: 0.8210 - val_loss: 0.7168 - val_accuracy: 0.7789
Epoch 18/50
1250/1250 [==============================] - 32s 26ms/step - loss: 0.4995 - accuracy: 0.8247 - val_loss: 0.6449 - val_accuracy: 0.7994
Epoch 19/50
1250/1250 [==============================] - 32s 26ms/step - loss: 0.4825 - accuracy: 0.8306 - val_loss: 0.6806 - val_accuracy: 0.7885
Epoch 20/50
1250/1250 [==============================] - 31s 25ms/step - loss: 0.4766 - accuracy: 0.8324 - val_loss: 0.6024 - val_accuracy: 0.8064
Epoch 21/50
1250/1250 [==============================] - 32s 26ms/step - loss: 0.4694 - accuracy: 0.8346 - val_loss: 0.6003 - val_accuracy: 0.8037
Epoch 22/50
1250/1250 [==============================] - 32s 25ms/step - loss: 0.4589 - accuracy: 0.8404 - val_loss: 0.6151 - val_accuracy: 0.8114
Epoch 23/50
1250/1250 [==============================] - 31s 25ms/step - loss: 0.4537 - accuracy: 0.8411 - val_loss: 0.6493 - val_accuracy: 0.8030
Epoch 24/50
1250/1250 [==============================] - 32s 25ms/step - loss: 0.4416 - accuracy: 0.8450 - val_loss: 0.6413 - val_accuracy: 0.8023
Epoch 25/50
1250/1250 [==============================] - 31s 25ms/step - loss: 0.4409 - accuracy: 0.8459 - val_loss: 0.6728 - val_accuracy: 0.7989
Epoch 26/50
1250/1250 [==============================] - 31s 25ms/step - loss: 0.4316 - accuracy: 0.8497 - val_loss: 0.6427 - val_accuracy: 0.8062
Epoch 27/50
1250/1250 [==============================] - 32s 25ms/step - loss: 0.4231 - accuracy: 0.8516 - val_loss: 0.6349 - val_accuracy: 0.8099
Epoch 28/50
1250/1250 [==============================] - 32s 25ms/step - loss: 0.4128 - accuracy: 0.8549 - val_loss: 0.6356 - val_accuracy: 0.8135
Epoch 29/50
1250/1250 [==============================] - 31s 25ms/step - loss: 0.4146 - accuracy: 0.8535 - val_loss: 0.6851 - val_accuracy: 0.7973
```

```
# Evaluate the model on the test set
test_loss2, test_acc2 = model2.evaluate(x_test, y_test)
print(f"Model 2 - Test accuracy: {test_acc2}")
print(f"Model 2 - Test loss: {test_loss2}")
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.7483 - accuracy: 0.8038
Model 2 - Test accuracy: 0.8037999868392944
Model 2 - Test loss: 0.748344898223877
```

```
# Plot training and validation loss for all epochs
plt.plot(history2.history['loss'], label='Training loss')
plt.plot(history2.history['val_loss'], label='Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Observations:

In the first model, validation loss increased by a lot after a few epochs. The model is also overfitting. There is a pattern where the training loss decreases proportionally while the validation loss increases, implying that the model is learning the training data well but failing to generalize to unfamiliar data.

In the second model, validation loss was stable and lower compared to the first model. This model is also less overfitting compared to the first model. Data augmentation definitely helps the model generalize better by introducing more variability in the training data, which improves its performance on the validation set.

```
# Model 3: CNN with Batch Normalization
model3 = Sequential([
    # First convolutional block
    Conv2D(32, (3, 3), padding='same', use_bias=False, input_shape=(32, 32, 3)),
    BatchNormalization(),
    Activation('relu'),
    Conv2D(32, (3, 3), padding='same', use_bias=False),
    BatchNormalization(),
    Activation('relu'),
    MaxPooling2D((2, 2)),

    # Second convolutional block
    Conv2D(64, (3, 3), padding='same', use_bias=False),
    BatchNormalization(),
    Activation('relu'),
    Conv2D(64, (3, 3), padding='same', use_bias=False),
    BatchNormalization(),
    Activation('relu'),
    MaxPooling2D((2, 2)),

    # Flattening layer
    Flatten(),

    # Fully connected layer
    Dense(512, use_bias=False),
    BatchNormalization(),
    Activation('relu'),

    # Output layer
    Dense(10, activation='softmax')
])
```

```
# Compile the model
model3.compile(optimizer=Adam(learning_rate=0.01), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Set up model checkpoint to save the best model
checkpoint3 = ModelCheckpoint('best_model3.h5', monitor='val_loss', save_best_only=True, mode='min')
```

```
# Train the model
history3 = model3.fit(x_train, y_train, epochs=50, batch_size=64, validation_data=(x_val, y_val), callbacks=[checkpoint3])
```

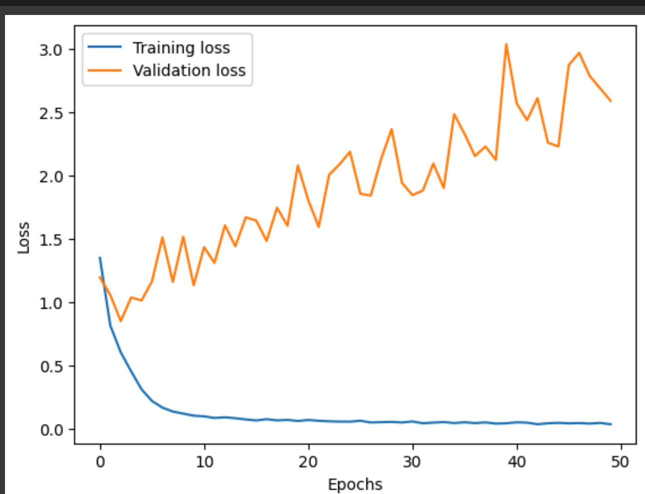
```
Epoch 1/50
625/625 [==============================] - 11s 12ms/step - loss: 1.3481 - accuracy: 0.5257 - val_loss: 1.1936 - val_accuracy: 0.5969
Epoch 2/50
625/625 [==============================] - 5s 9ms/step - loss: 0.8129 - accuracy: 0.7150 - val_loss: 1.0545 - val_accuracy: 0.6494
```

```
Epoch 3/50
625/625 [==============================] - 6s 9ms/step - loss: 0.6019 - accuracy: 0.7886 - val_loss: 0.8481 - val_accuracy: 0.7263
Epoch 4/50
625/625 [==============================] - 5s 9ms/step - loss: 0.4514 - accuracy: 0.8403 - val_loss: 1.0356 - val_accuracy: 0.6858
Epoch 5/50
625/625 [==============================] - 6s 10ms/step - loss: 0.3092 - accuracy: 0.8900 - val_loss: 1.0115 - val_accuracy: 0.7159
Epoch 6/50
625/625 [==============================] - 5s 8ms/step - loss: 0.2173 - accuracy: 0.9234 - val_loss: 1.1635 - val_accuracy: 0.7239
Epoch 7/50
625/625 [==============================] - 5s 9ms/step - loss: 0.1647 - accuracy: 0.9428 - val_loss: 1.5090 - val_accuracy: 0.6831
Epoch 8/50
625/625 [==============================] - 6s 9ms/step - loss: 0.1339 - accuracy: 0.9543 - val_loss: 1.1587 - val_accuracy: 0.7309
Epoch 9/50
625/625 [==============================] - 5s 8ms/step - loss: 0.1177 - accuracy: 0.9589 - val_loss: 1.5152 - val_accuracy: 0.7119
Epoch 10/50
625/625 [==============================] - 6s 9ms/step - loss: 0.1011 - accuracy: 0.9656 - val_loss: 1.1317 - val_accuracy: 0.7578
Epoch 11/50
625/625 [==============================] - 5s 8ms/step - loss: 0.0962 - accuracy: 0.9667 - val_loss: 1.4323 - val_accuracy: 0.7269
Epoch 12/50
625/625 [==============================] - 6s 9ms/step - loss: 0.0831 - accuracy: 0.9718 - val_loss: 1.3081 - val_accuracy: 0.7609
Epoch 13/50
625/625 [==============================] - 5s 8ms/step - loss: 0.0884 - accuracy: 0.9707 - val_loss: 1.6050 - val_accuracy: 0.7294
Epoch 14/50
625/625 [==============================] - 5s 8ms/step - loss: 0.0811 - accuracy: 0.9719 - val_loss: 1.4395 - val_accuracy: 0.7587
Epoch 15/50
625/625 [==============================] - 6s 9ms/step - loss: 0.0717 - accuracy: 0.9763 - val_loss: 1.6686 - val_accuracy: 0.7275
Epoch 16/50
625/625 [==============================] - 5s 8ms/step - loss: 0.0636 - accuracy: 0.9798 - val_loss: 1.6428 - val_accuracy: 0.7410
Epoch 17/50
625/625 [==============================] - 6s 9ms/step - loss: 0.0736 - accuracy: 0.9758 - val_loss: 1.4821 - val_accuracy: 0.7636
Epoch 18/50
625/625 [==============================] - 5s 8ms/step - loss: 0.0642 - accuracy: 0.9788 - val_loss: 1.7440 - val_accuracy: 0.7476
Epoch 19/50
625/625 [==============================] - 6s 9ms/step - loss: 0.0680 - accuracy: 0.9785 - val_loss: 1.6015 - val_accuracy: 0.7435
Epoch 20/50
625/625 [==============================] - 5s 8ms/step - loss: 0.0595 - accuracy: 0.9805 - val_loss: 2.0777 - val_accuracy: 0.7219
Epoch 21/50
625/625 [==============================] - 5s 8ms/step - loss: 0.0673 - accuracy: 0.9782 - val_loss: 1.8033 - val_accuracy: 0.7622
Epoch 22/50
625/625 [==============================] - 6s 9ms/step - loss: 0.0610 - accuracy: 0.9806 - val_loss: 1.5925 - val_accuracy: 0.7655
Epoch 23/50
625/625 [==============================] - 5s 8ms/step - loss: 0.0568 - accuracy: 0.9823 - val_loss: 2.0046 - val_accuracy: 0.7467
Epoch 24/50
625/625 [==============================] - 6s 9ms/step - loss: 0.0547 - accuracy: 0.9836 - val_loss: 2.0870 - val_accuracy: 0.7467
Epoch 25/50
625/625 [==============================] - 5s 8ms/step - loss: 0.0541 - accuracy: 0.9828 - val_loss: 2.1857 - val_accuracy: 0.7372
Epoch 26/50
625/625 [==============================] - 5s 9ms/step - loss: 0.0611 - accuracy: 0.9820 - val_loss: 1.8543 - val_accuracy: 0.7641
Epoch 27/50
625/625 [==============================] - 5s 9ms/step - loss: 0.0475 - accuracy: 0.9849 - val_loss: 1.8404 - val_accuracy: 0.7723
Epoch 28/50
625/625 [==============================] - 5s 8ms/step - loss: 0.0501 - accuracy: 0.9846 - val_loss: 2.1317 - val_accuracy: 0.7405
Epoch 29/50
625/625 [==============================] - 6s 9ms/step - loss: 0.0520 - accuracy: 0.9835 - val_loss: 2.3651 - val_accuracy: 0.7571
```

```
# Evaluate the model on the test set
test_loss3, test_acc3 = model3.evaluate(x_test, y_test)
print(f"Model 3 - Test accuracy: {test_acc3}")
print(f"Model 3 - Test loss: {test_loss3}")
```

```
313/313 [==============================] - 2s 4ms/step - loss: 2.5398 - accuracy: 0.7631
Model 3 - Test accuracy: 0.7631000280380249
Model 3 - Test loss: 2.5397677421569824
```

```
# Plot training and validation loss for all epochs
plt.plot(history3.history['loss'], label='Training loss')
plt.plot(history3.history['val_loss'], label='Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Observations:

In the third model, the validation loss is high and fluctuating. The batch normalization helps stabilize the training. However, the model is still overfitting.

https://colab.research.google.com/drive/1QBdZnFcRcZOUDcGEUX5OTiA99pnyOHjg?usp=sharing

```
Start coding or generate with AI.
```