CISC 867 Project 2: The Fashion-MNIST clothing classification dataset using CNN and transfer learning

By:

Alhassan Ehab Ramadan Ahmed

ID:

20398553

Supervised by:

Prof. Hazem Abbas

Project objective:

- In this project, the Fashion-MNIST clothing classification was used by CNN architecture and transfer learing**.**

Problem Description:

- **input**: 28×28-pixel grayscale images of items of 10 types of clothing, such as shoes, t-shirts, dresses, and more
- **Output**: Predict the type of clothes (Bag, Shirt ...etc.)
- **Challenges**: Interact with Transfer learning and choose the best hyperparameters for the network

Dataset Description:

The Fashion-MNIST dataset consists of images of that originate from Zalando's image directory. Zalando is a European e-commerce company founded in 2008. The researchers in Zalando have created the Fashion-MNIST dataset that contains 70,000 images of clothing. More specifically, it contains 60,000 training examples and 10,000 testing examples, which are all grayscale images with the dimension 28 x 28 categorized into 10 classes. It is a dataset comprised of 60,000 square (28×28) grayscale images of items of 10 types of clothing, such as shoes, t-shirts, dresses, and more. The mapping of all 0-9 integers to class labels is listed below

0: T-shirt/top
1: Trouser
2: Pullover
3: Dress
4: Coat
5: Sandal
6: Shirt
7: Sneaker
8: Bag
9: Ankle bo

Part I: Data Preparation

Import libraries:

```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from keras.models import Sequential
from keras.layers import Conv2D, Activation, MaxPool2D, Flatten, Dense, GlobalAveragePooling2D, Dropout
from sklearn.model_selection import KFold, train_test_split
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
from tensorflow import keras
from kerastuner import RandomSearch
from kerastuner.engine.hyperparameters import HyperParameters
```

Read the data:

```
#load train data
df = pd.read_csv('/kaggle/input/fashionmnist/fashion-mnist_train.csv')
df.head()
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783	pixel784
0	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	6	0	0	0	0	0	0	0	5	0	...	0	0	0	30	43	0	0	0	0	0
3	0	0	0	0	1	2	0	0	0	0	...	3	0	0	0	0	1	0	0	0	0
4	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows x 785 columns

```
#print dimension of the data
df.shape
```

```
(60000, 785)
```

Check nulls and drop duplicates:

```
#check missing values
df.isna().sum().sort_values()
```

```
label       0
pixel517    0
pixel518    0
pixel519    0
pixel520    0
           ..
pixel264    0
pixel265    0
pixel266    0
pixel268    0
pixel784    0
Length: 785, dtype: int64
```

```
#check duplicated data
df.duplicated().any()
```

```
True
```

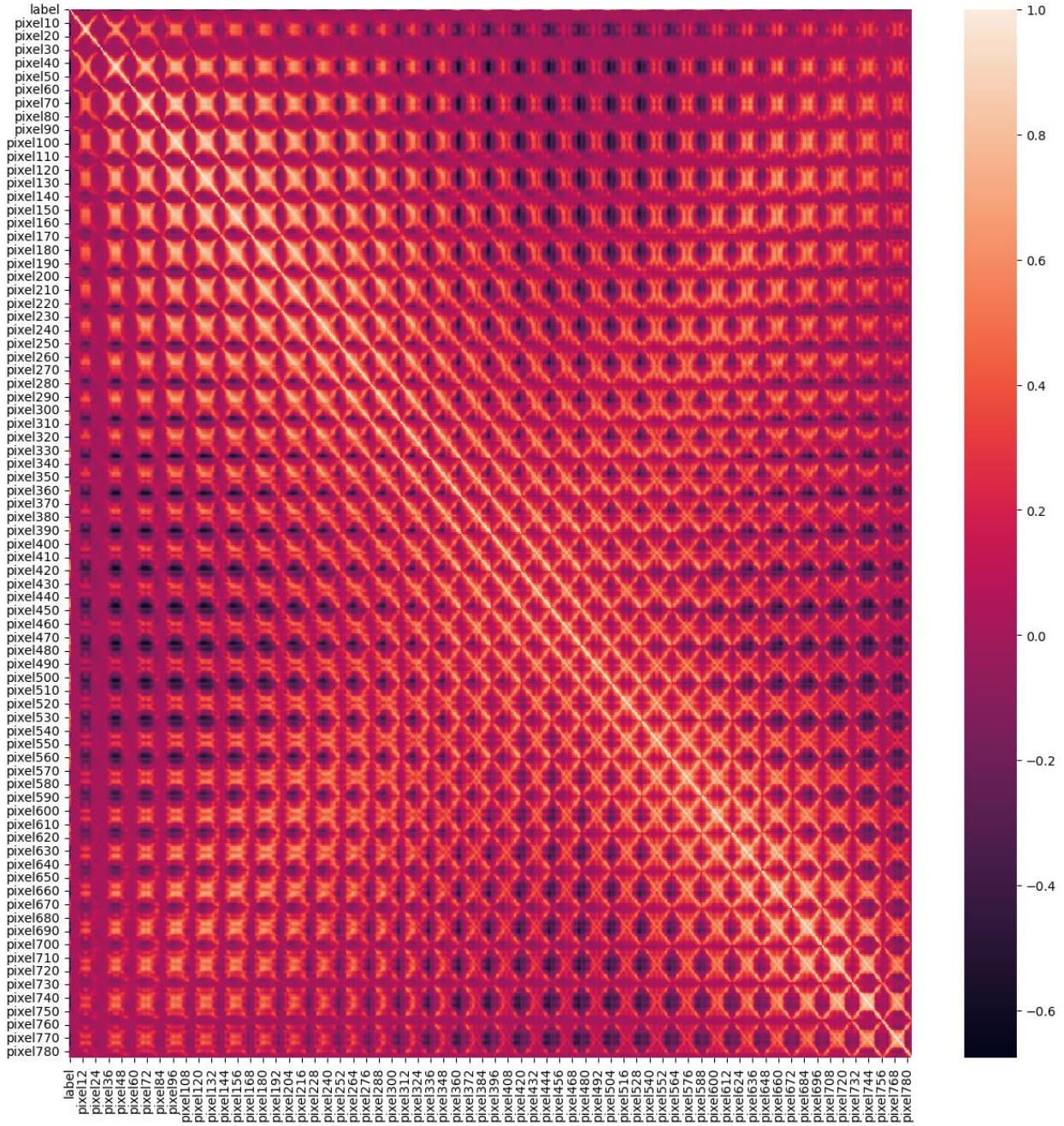
```
#display number of duplicated data
df.duplicated().sum()
```

```
43
```

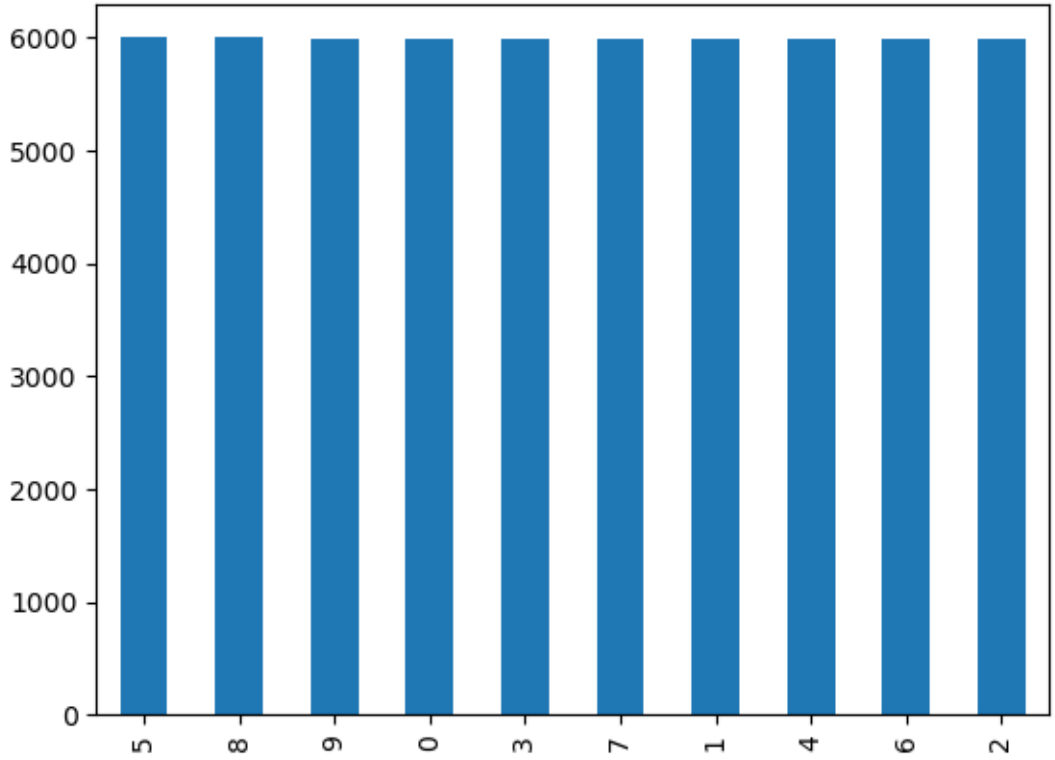
```
#drop duplicates
df = df.drop_duplicates()
df.shape
```

```
(59957, 785)
```

Data visualization (heatmap) :



Number of labels for each class on training data:



Drawing some images:

Dataset:

- assigned target column to y and encoded classes
- assign pixels to X
- normalize pixels value
- split the data into train and validation
- print splits dimension

```
[ ] #assign the label to y variable and x to the pixels
    y = df["label"]
    X = df.drop(["label"], axis =1)
```

```
#normalize image pixels and encode its label
X_train = (X.values.reshape(59957,28,28,1))/255
y_train = to_categorical(y)
```

```
[ ] #split data into tain and validation
    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

    #print splits dimension
    print(X_train.shape)
    print(X_val.shape)
    print(y_train.shape)
    print(y_val.shape)
```

```
(47965, 28, 28, 1)
(11992, 28, 28, 1)
(47965, 10)
(11992, 10)
```

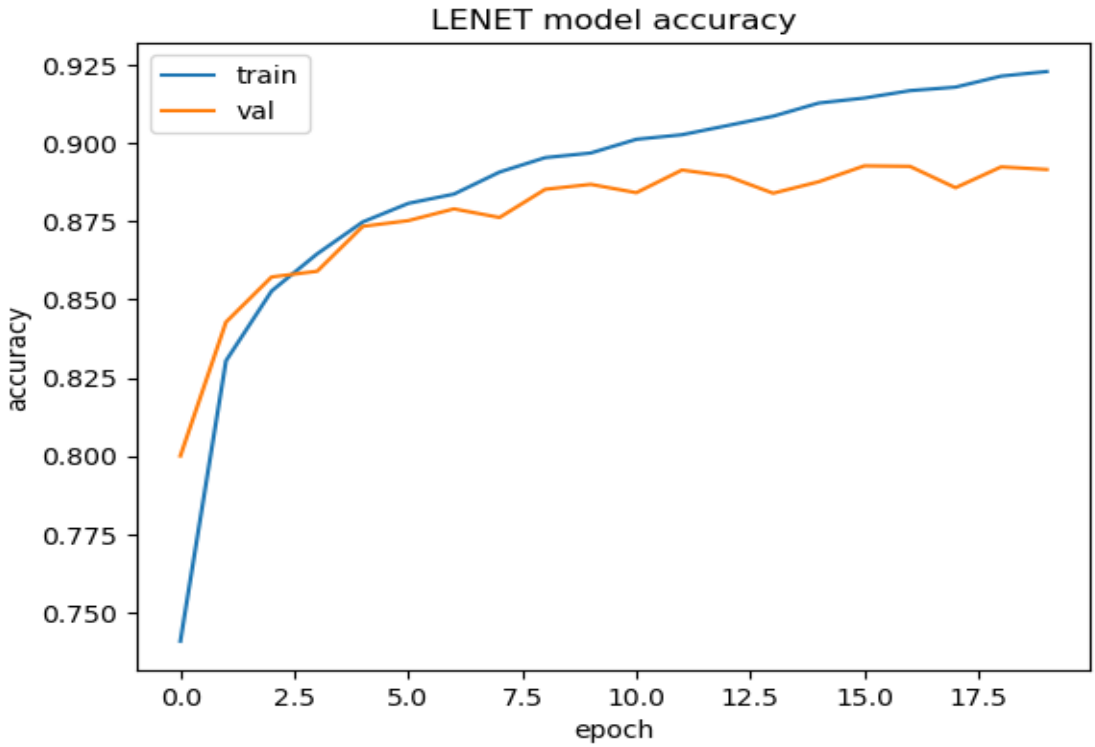
Part II: Training a CNN neural network

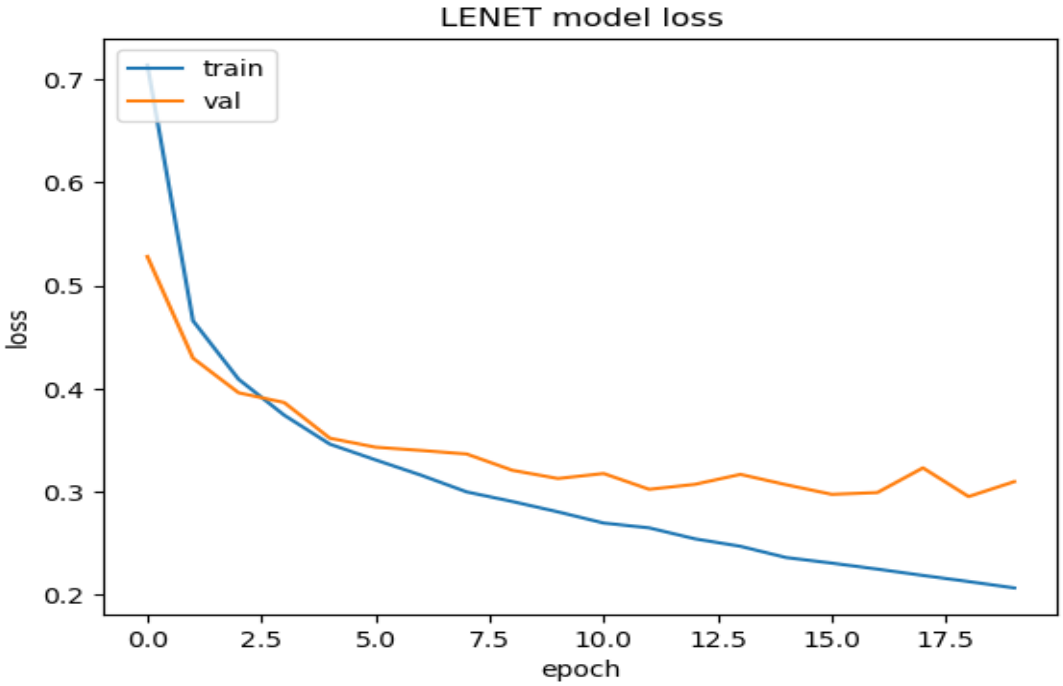
LeNet-5 model architecture:

```
#show model summery
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 24, 24, 6)         156

 max_pooling2d (MaxPooling2D  (None, 12, 12, 6)        0
 )

 conv2d_1 (Conv2D)           (None, 8, 8, 16)          2416

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 16)         0
 2D)

 flatten_7 (Flatten)         (None, 256)               0

 dense_19 (Dense)            (None, 120)               30840

 dense_20 (Dense)            (None, 84)                10164

 dense_21 (Dense)            (None, 10)                850

=================================================================
Total params: 44,426
Trainable params: 44,426
Non-trainable params: 0
_________________________________________________________________
```

Plot accuracy and loss for the LeNet-5 model:





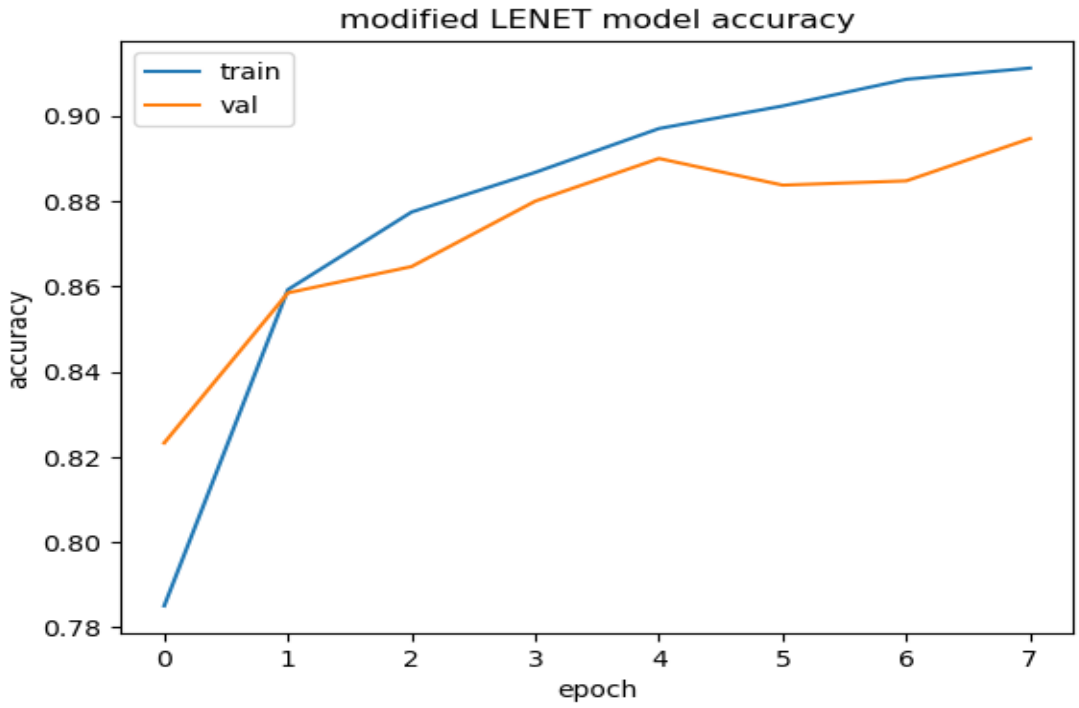
Modified LeNet-5 model architecture:

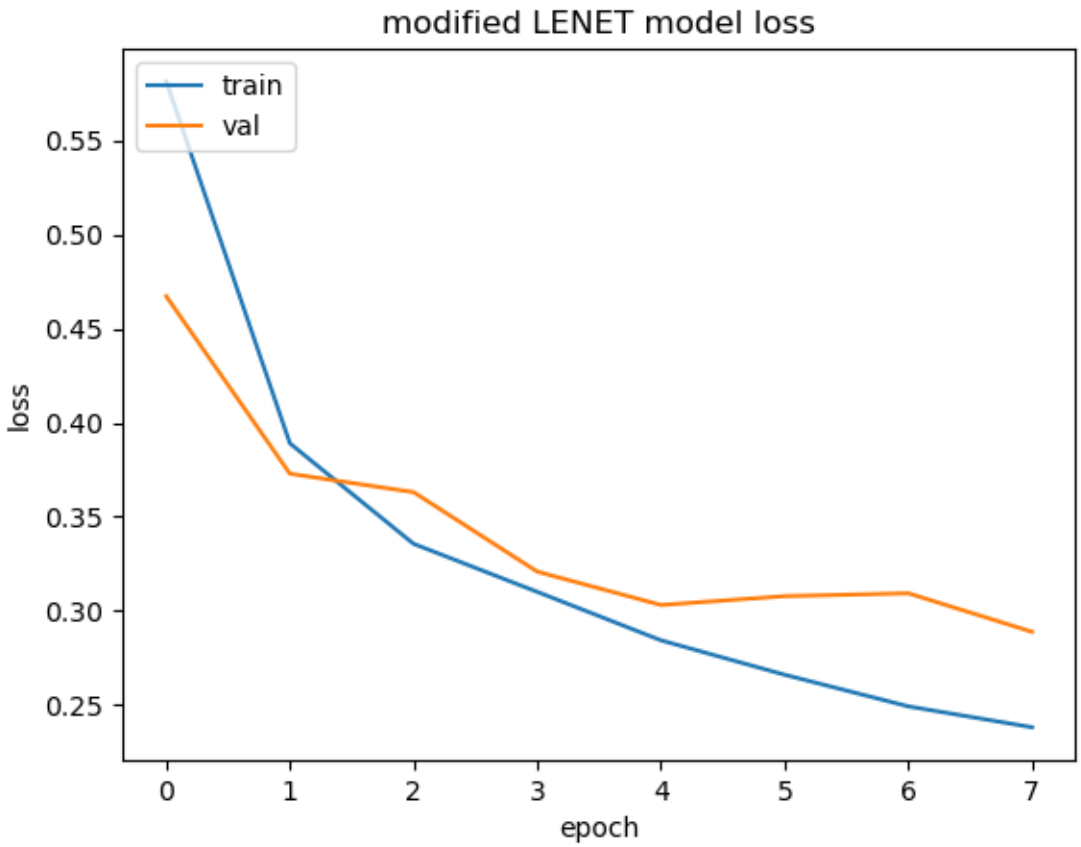
```
[ ]  #build a function which has lenet model architecture and tuner trying to achive best performance
     # tuner try to tune the number of nurens in the first dense layer and values of learning rat
     def build_model(hp):
         modell = Sequential()
         modell.add(Conv2D(6, (5,5),activation='relu',input_shape=(28,28,1)))
         modell.add(MaxPool2D((2,2)))
         modell.add(Conv2D(16, (5,5), activation='relu'))
         modell.add(MaxPool2D((2,2)))
         modell.add(Flatten())

         modell.add(Dense(units=hp.Int('dense_units', min_value=64, max_value=128, step=16),activation='relu'))
         modell.add(Dense(84, activation='relu'))
         modell.add(Dense(10, activation='softmax'))

         modell.compile(optimizer=keras.optimizers.Adam(hp.Choice('learning_rate', values=[1e-2, 1e-3])),
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])
         return modell
```

As shown in the above figure I used the tuner to tune the hyperparameters in this model trying to get the best performance and I applied the tuner on the number of neurons in the dense layer and on the learning rate.

It was observed that the accuracy for training and validation slightly increased and loss for training and validation slightly decreased

Plot accuracy and loss for the LeNet-5 model:





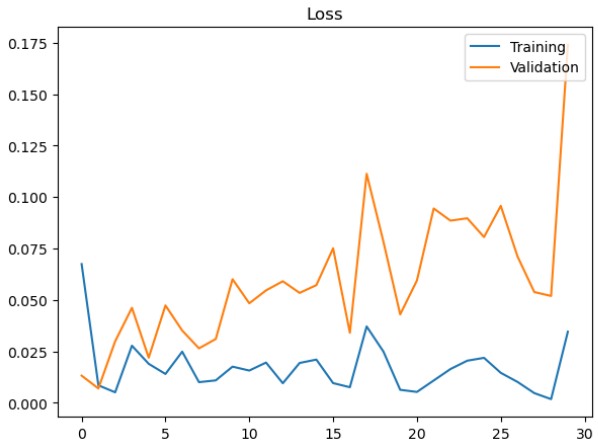
Observation:

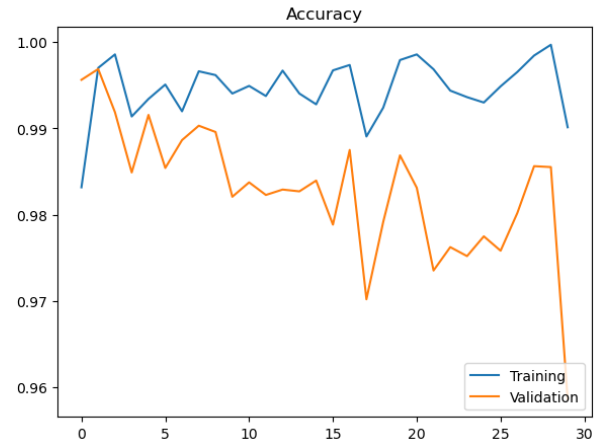
At the first, I tried 35 epochs but I noticed that the model overfits after 8 epochs so I performed 8 epochs.

Cross-validation:

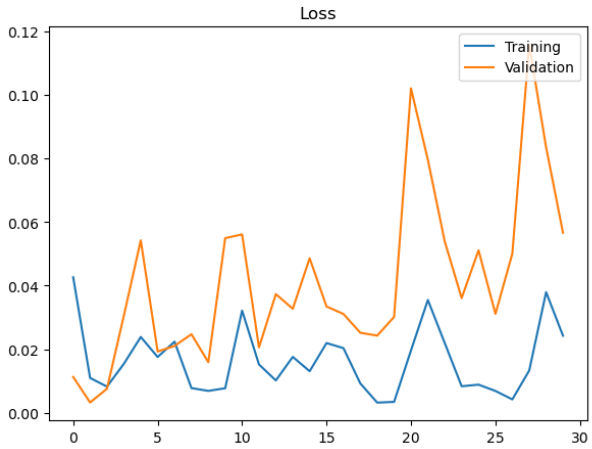
At this part, I specified the number of folds with 5 and performed cross-validation on the modified LeNet-5 model and it was observed that fold 4 was the best regarding accuracy and loss. In addition, the other four folds fluctuate through the 30 epochs while fold 4 starts with fluctuates till epoch 10 and then tends to be stable. The figures will be shown below

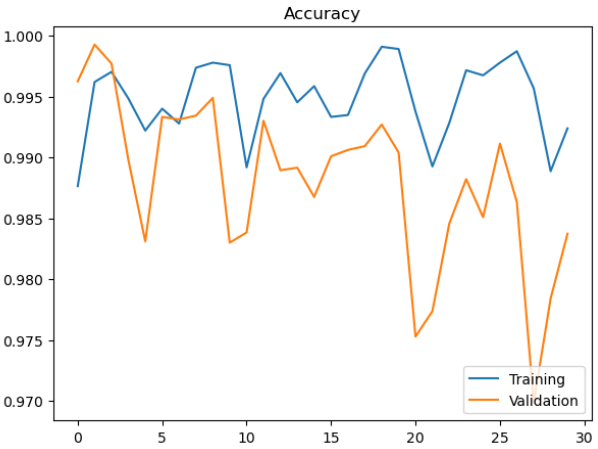
Fold-1



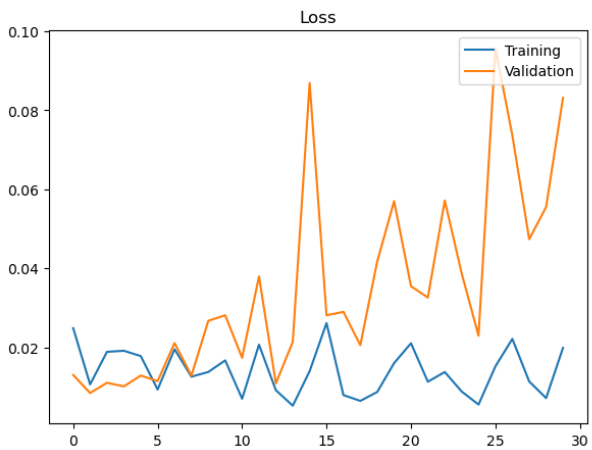


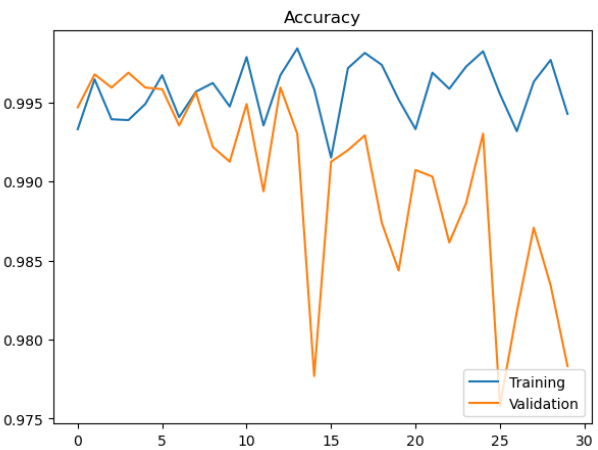
Fold-2

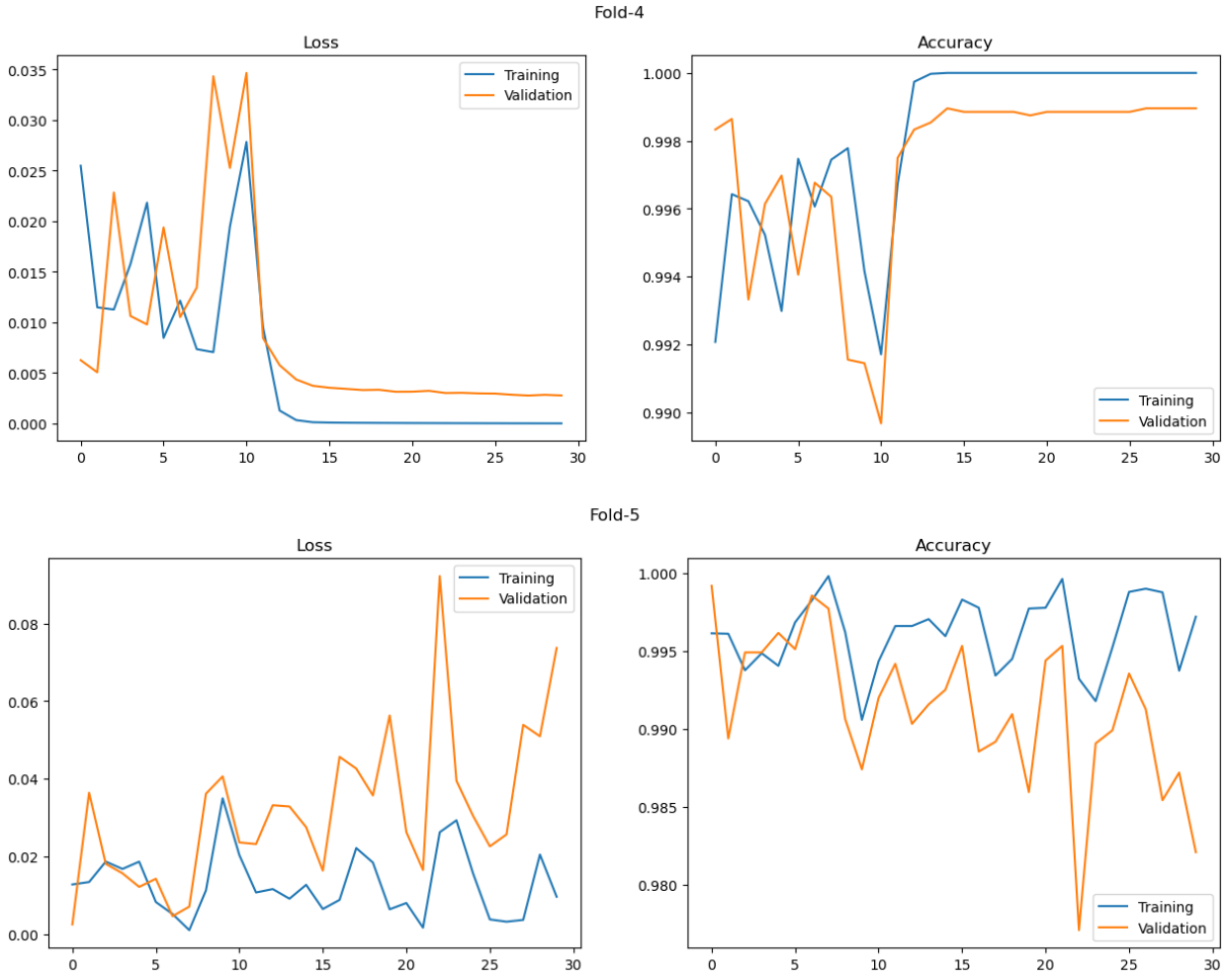




Fold-3



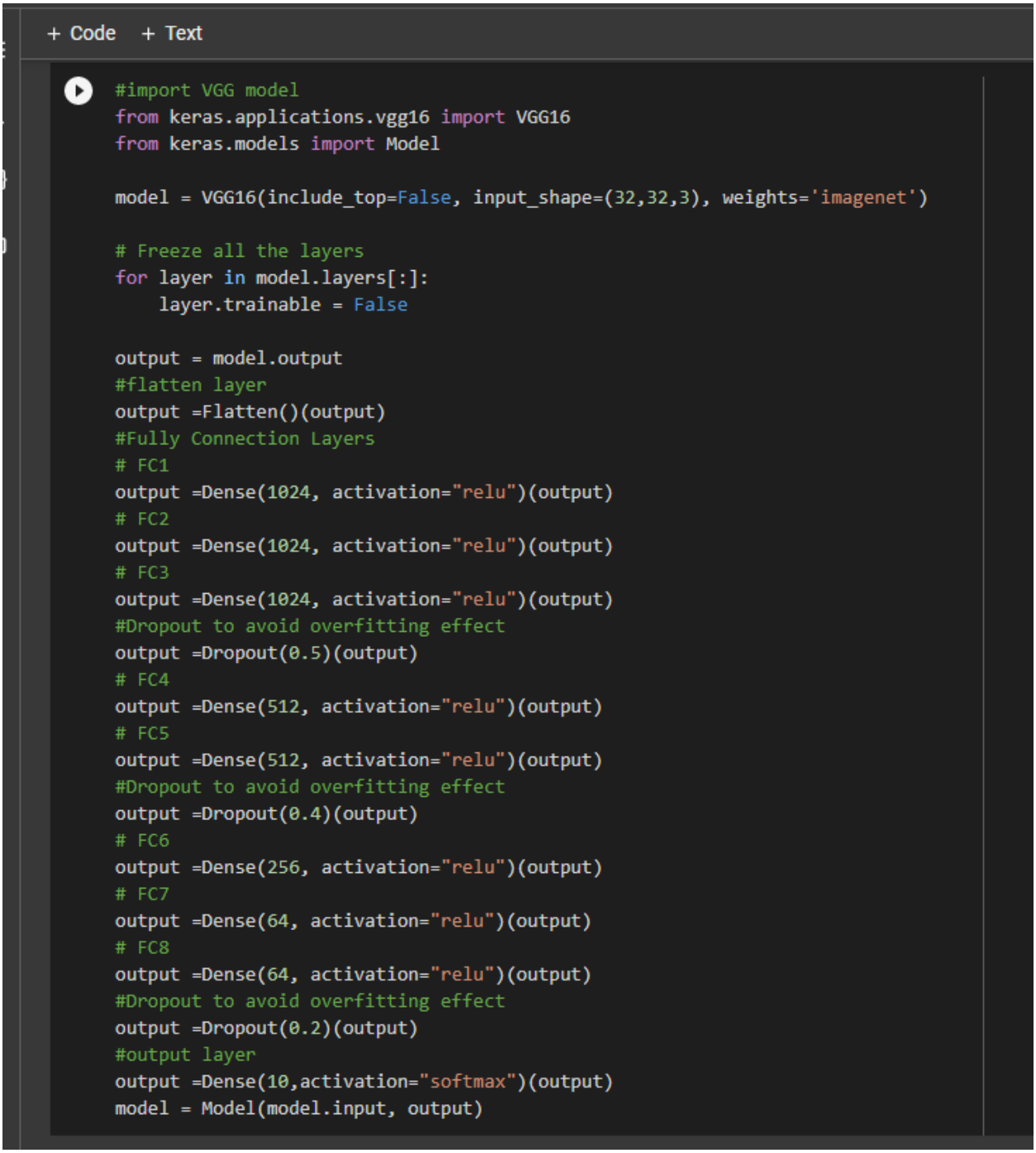




Comment on why you think LeNet-5 further improves the accuracy if any at all. And if it doesn't, why not?

It is unlikely that LeNet-5 would further improve accuracy on modern datasets because it was designed more than two decades ago when were no GPUs and memory, in addition, its architecture is relatively simple consists of only 2 convolutional layers therefore it will be difficult to extract more features compared to current state-of-the-art CNNs.

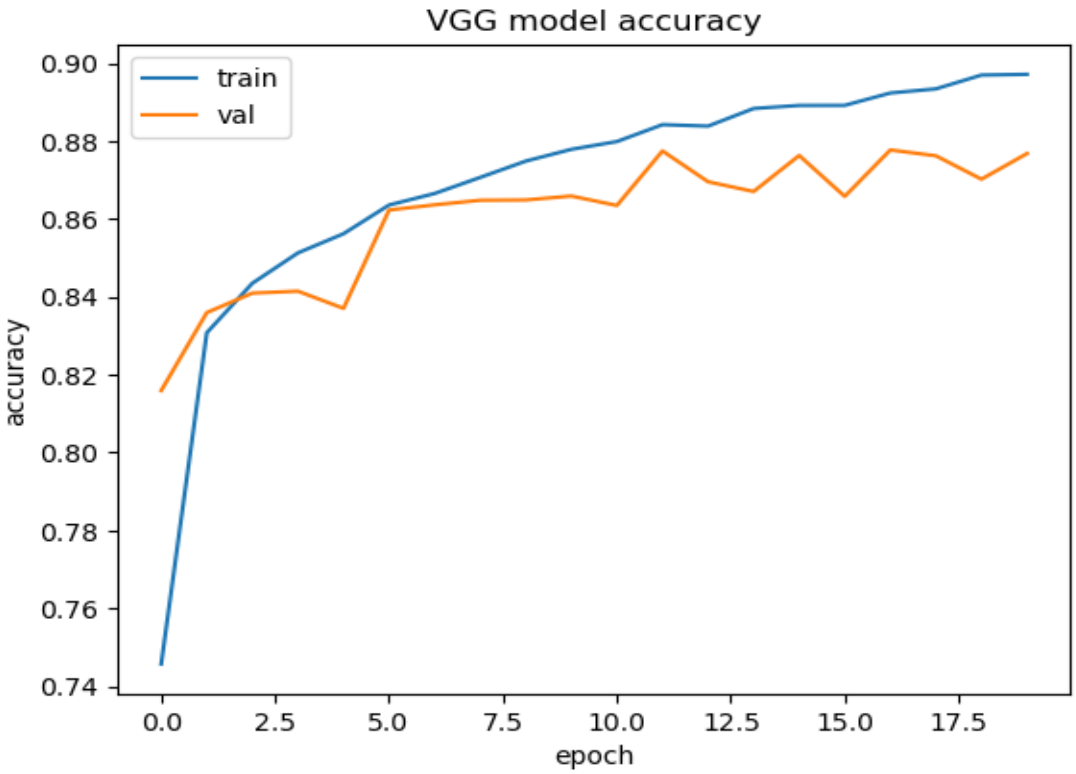
VGG-16 model architecture:

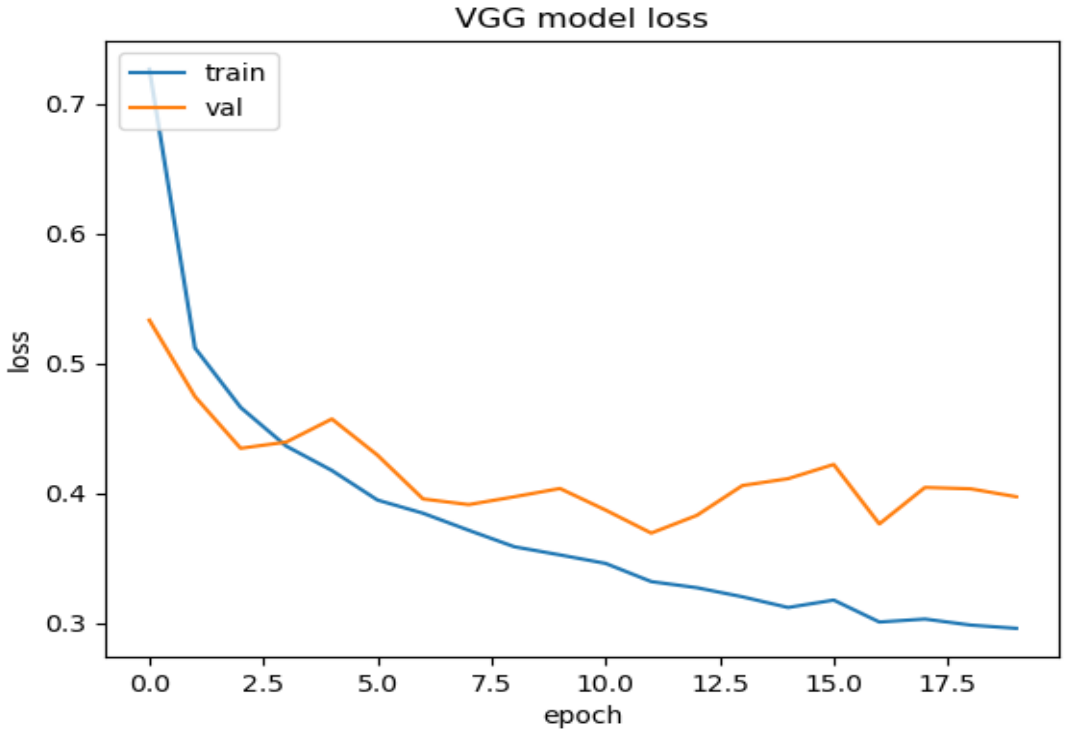


+ Code + Text

```
#import VGG model
from keras.applications.vgg16 import VGG16
from keras.models import Model

model = VGG16(include_top=False, input_shape=(32,32,3), weights='imagenet')

# Freeze all the layers
for layer in model.layers[:]:
    layer.trainable = False

output = model.output
#flatten layer
output =Flatten()(output)
#Fully Connection Layers
# FC1
output =Dense(1024, activation="relu")(output)
# FC2
output =Dense(1024, activation="relu")(output)
# FC3
output =Dense(1024, activation="relu")(output)
#Dropout to avoid overfitting effect
output =Dropout(0.5)(output)
# FC4
output =Dense(512, activation="relu")(output)
# FC5
output =Dense(512, activation="relu")(output)
#Dropout to avoid overfitting effect
output =Dropout(0.4)(output)
# FC6
output =Dense(256, activation="relu")(output)
# FC7
output =Dense(64, activation="relu")(output)
# FC8
output =Dense(64, activation="relu")(output)
#Dropout to avoid overfitting effect
output =Dropout(0.2)(output)
#output layer
output =Dense(10,activation="softmax")(output)
model = Model(model.input, output)
```

Plot accuracy and loss for the VGG-16 model:

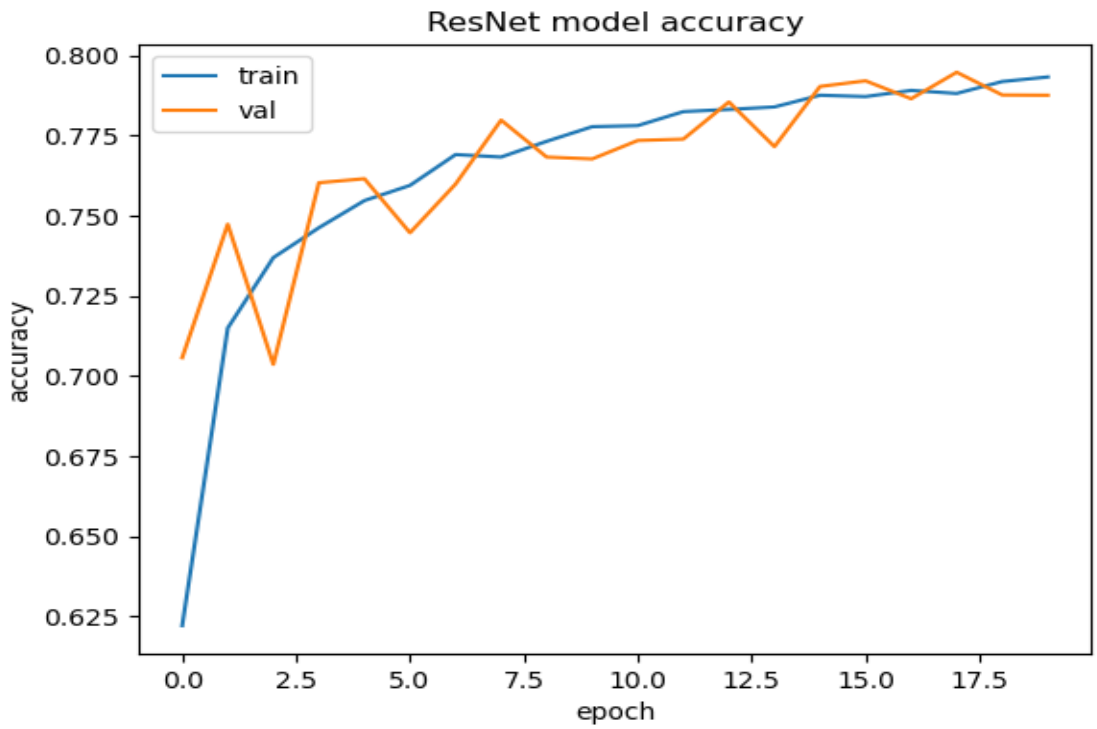


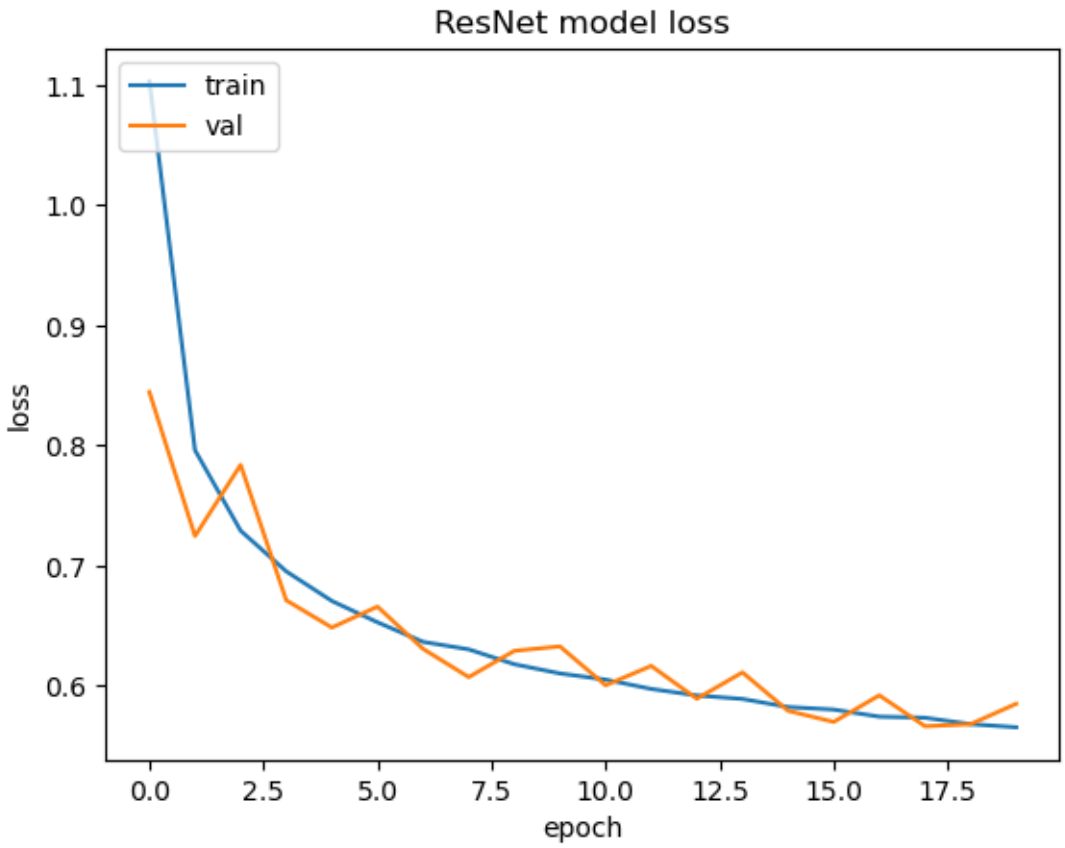


ResNet-50 model architecture:

```
#import ResNet50
from tensorflow.keras.applications.resnet50 import ResNet50
from keras.layers import  GlobalAveragePooling2D


model = ResNet50(include_top=False, input_shape=(32,32,3), weights='imagenet')

# Freeze all the layers
for layer in model.layers[:]:
    layer.trainable = False

# Add Dense layer
output = model.output
output = GlobalAveragePooling2D()(output)
output = Dense(units=10, activation='softmax')(output)
model = Model(model.input, output)
```

Plot accuracy and loss for the VGG-16 model:





Observations on VGG-16 and ResNet-50:

Before passing the images to the input layer, I had to resize the images into 32x32 and repeat single channel to three to be compatible with the minimum requirements for these two models.

Result Comparison between transfer learning models (VGG-16, ResNet-50) and full-trained model (LeNet-5):

	LeNet-5	VGG-16	ResNet-50
val_accuracy	0.8947	0.8768	0.7875
val_loss	0.2888	0.3973	0.5851

As shown in the above results LeNet-5 has a higher performance than VGG-16 and ResNet-50, and VGG-19 is better than ResNet-50

Summary:

It is possible for a fully trained CNN model like LeNet-5 to achieve higher accuracy than a pre-trained model such as VGG-16 and ResNet-50 in certain situations. This can occur when the pre-trained model is not well-suited to the specific task or dataset which occurred in our situation.

the data was not suitable for our transferred models and I had to perform some operations on it before passing it to the model and LeNet-5 was built from scratch therefore, its weights are updated well. the specific performance of each model can depend on several factors.