Fashion MNIST CNN Model Report

Introduction

This document summarizes the development and training of a Convolutional Neural Network (CNN) on the Fashion MNIST dataset using TensorFlow and Keras. .In this document we discuss the result of Alex_Net as well as some other well known convolution . We also discuss the results and effects of the convolutions on different classes.

More specifically, we discuss the classes of identifying Dogs vs Cat [2] and Fashion -MNIST [9].

The Fashion MNIST dataset contains grayscale images of clothing items, and the objective is to classify them into one of ten categories.

Data Preprocessing[For Fashion-MNIST]

The dataset was loaded and normalized by dividing pixel values by 255. Additionally, a new axis was added to match the input shape required for CNNs.

```
from tensorflow.keras.datasets import fashion_mnist
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

x_train_split, x_val_split, y_train_split, y_val_split = train_test_split(

   x_train, y_train, test_size=0.2, random_state=42

)

x_train = x_train.astype('float32') / 255.0
```

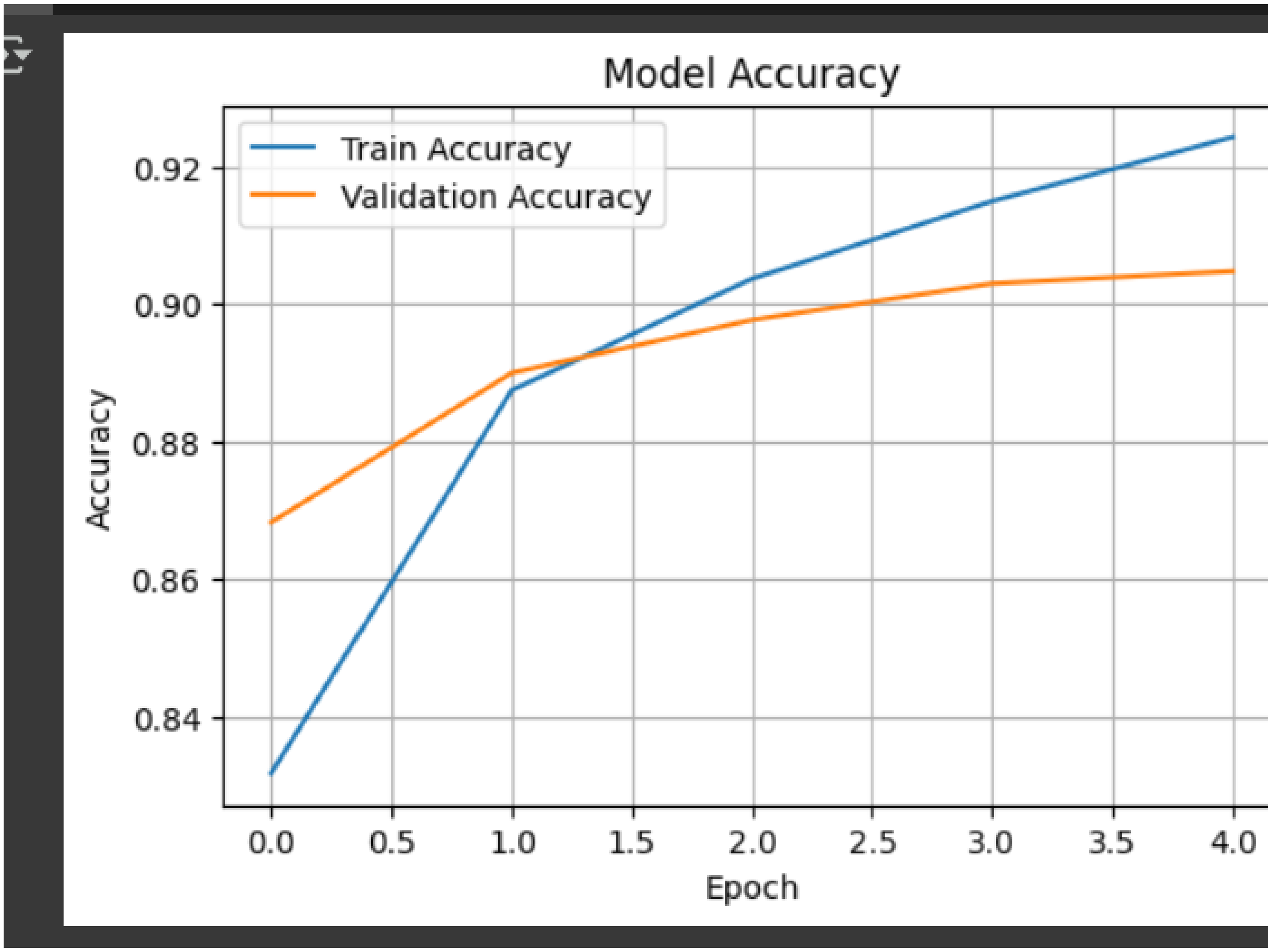
x_test = x_test.astype('float32') / 255.0

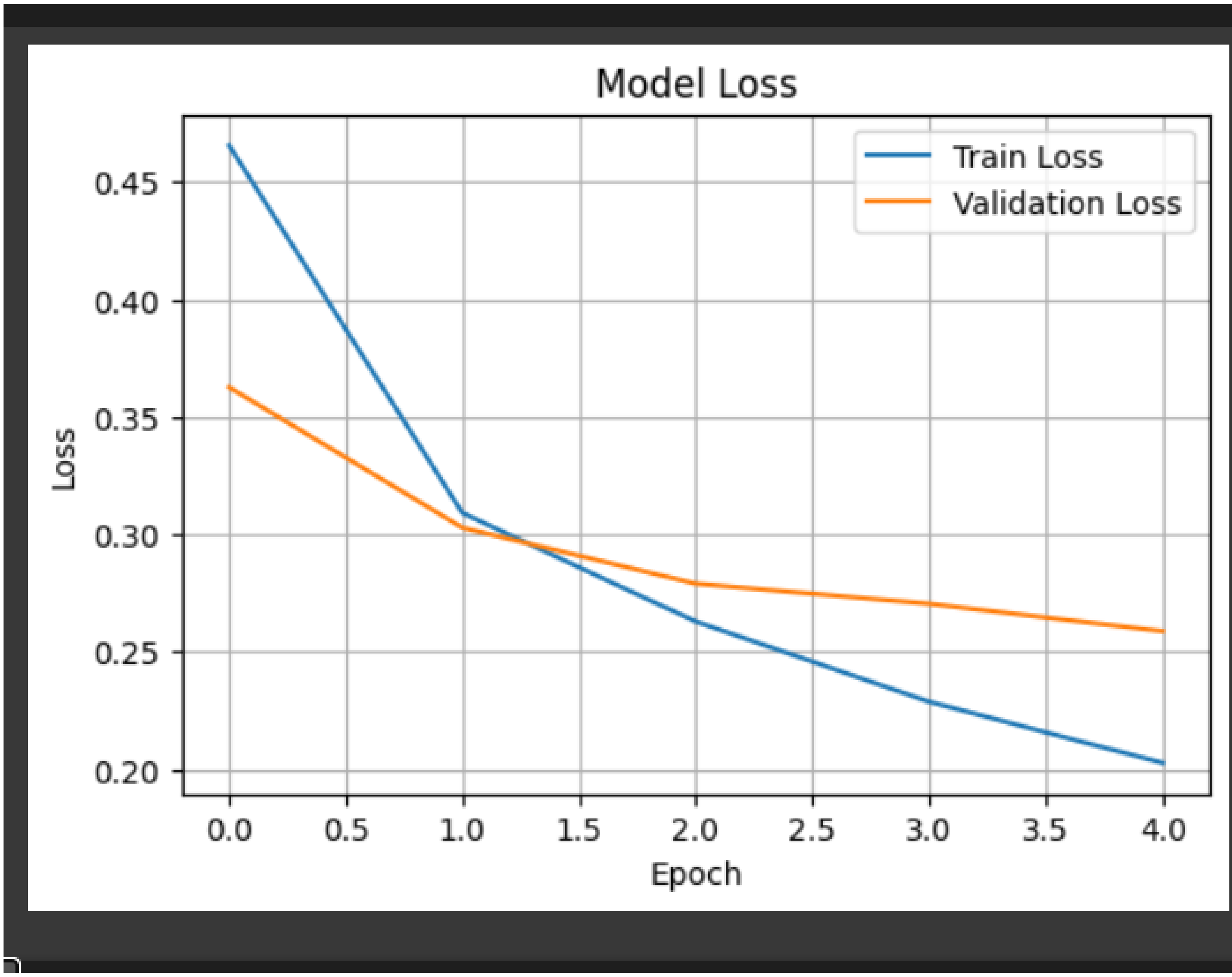
Model Architecture[Two cases]

First Case:

A sequential CNN model was used with the following layers:

```
model = tf.keras.Sequential([
   tf.keras.layers.Conv2D(32, (3,3), activation='relu',
input_shape=(28,28,1)),
   tf.keras.layers.MaxPooling2D(2,2),
   tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
   tf.keras.layers.MaxPooling2D(2,2),
   tf.keras.layers.Flatten(),
   tf.keras.layers.Dense(128, activation='relu'),
   tf.keras.layers.Dense(10, activation='softmax')  # 10 classes
])
```


Model Accuracy
Train Accuracy
Validation Accuracy
0.92
0.90
0.88
0.86
0.84
Accuracy
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epoch

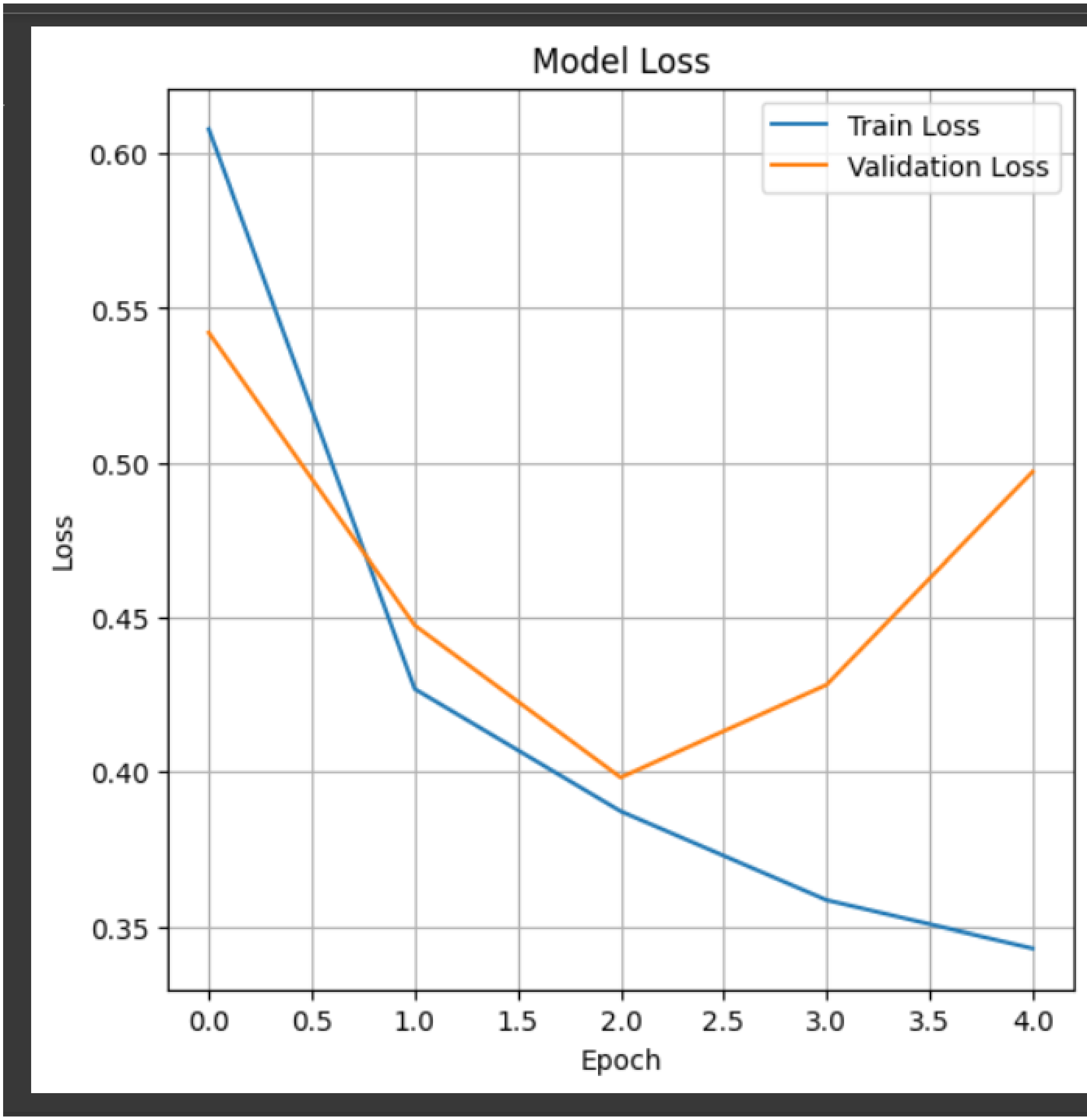


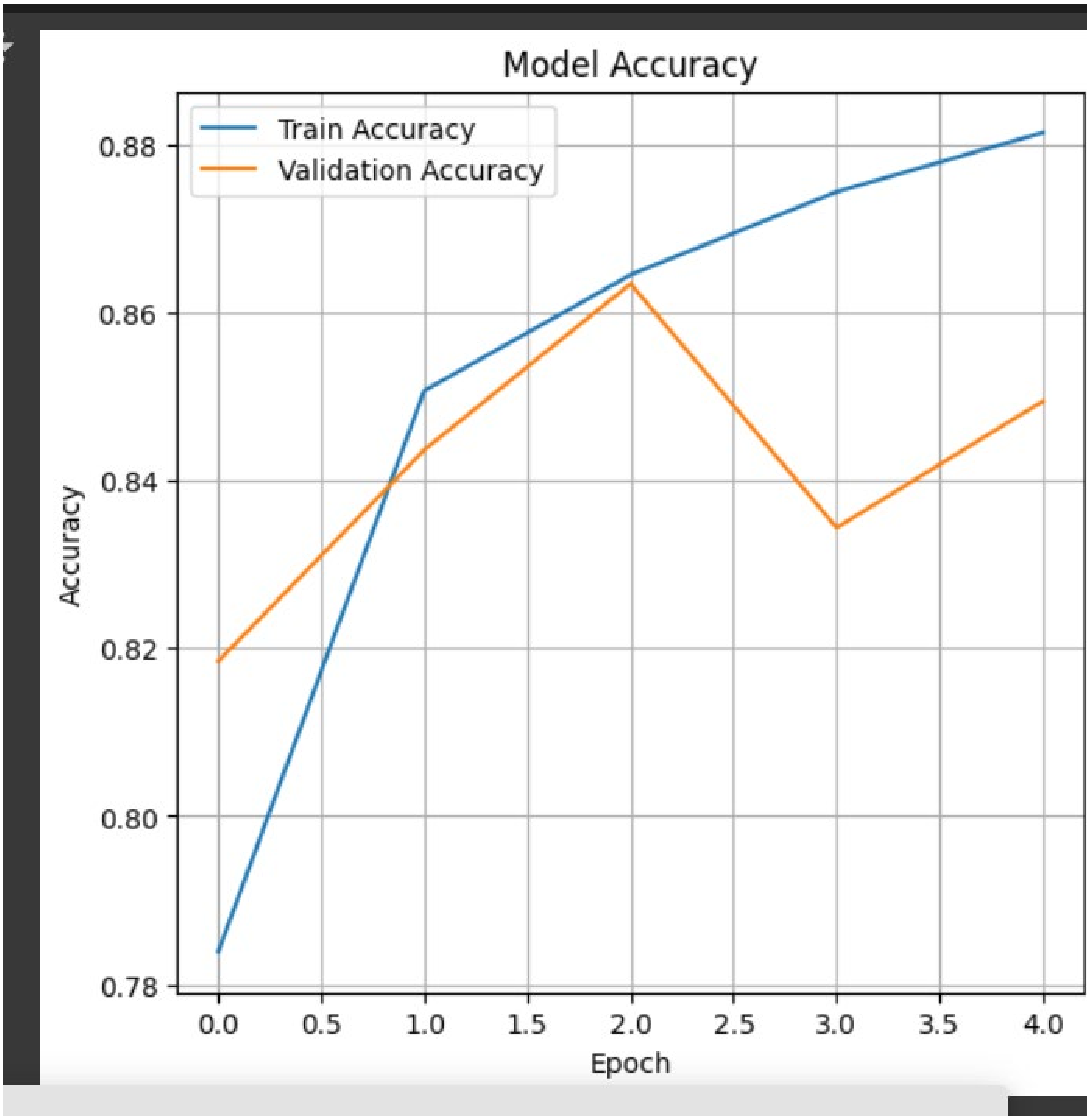
This actually worked really well , we see the loss steadily decreasing with each epoch and the final accuracy we see is >90% .Also the time taken to evaluate each row was low as well(<5ms)

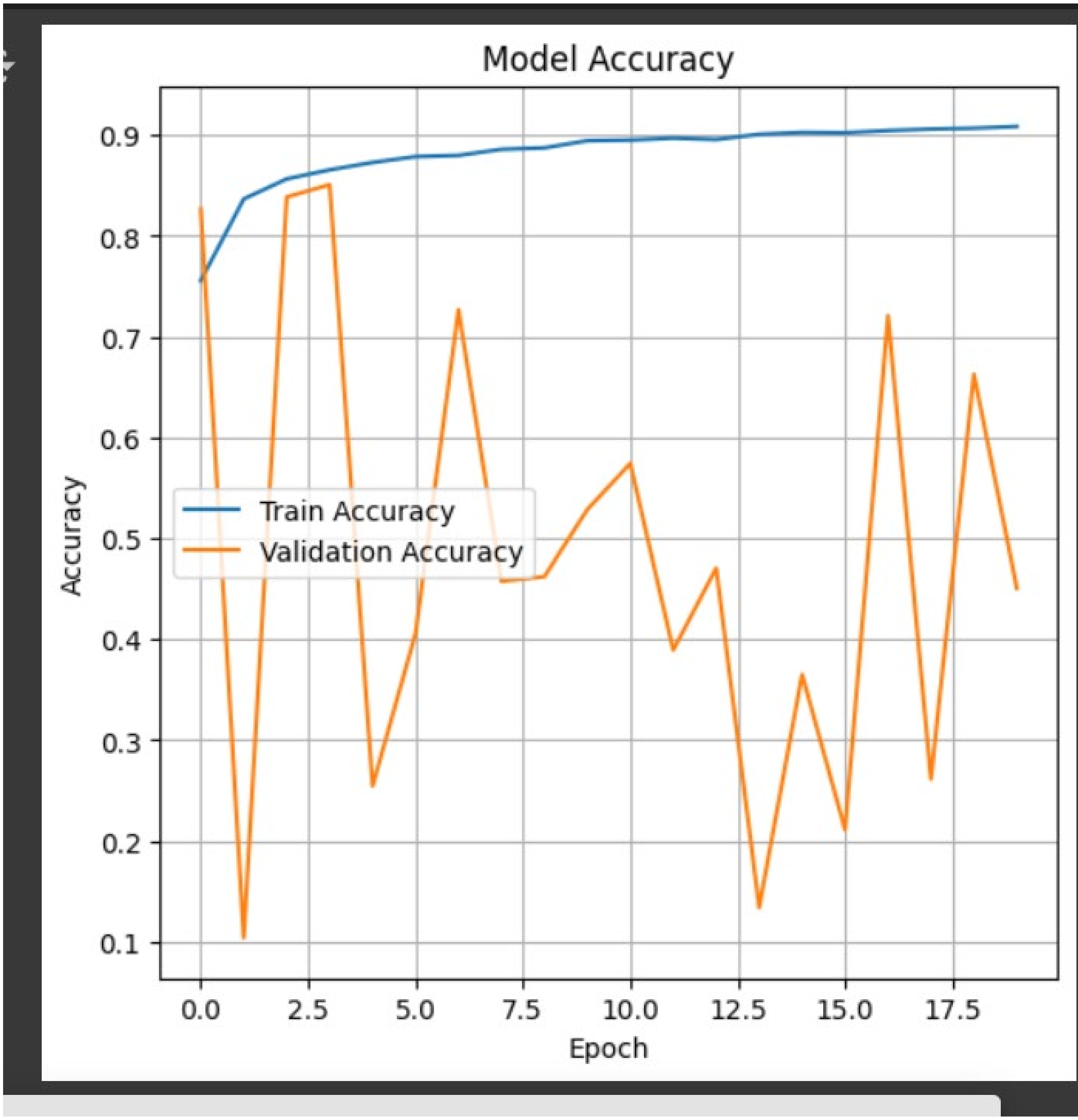
I presume the reason being that it is much simple , there are only ten layers and this works really well for greyscale images

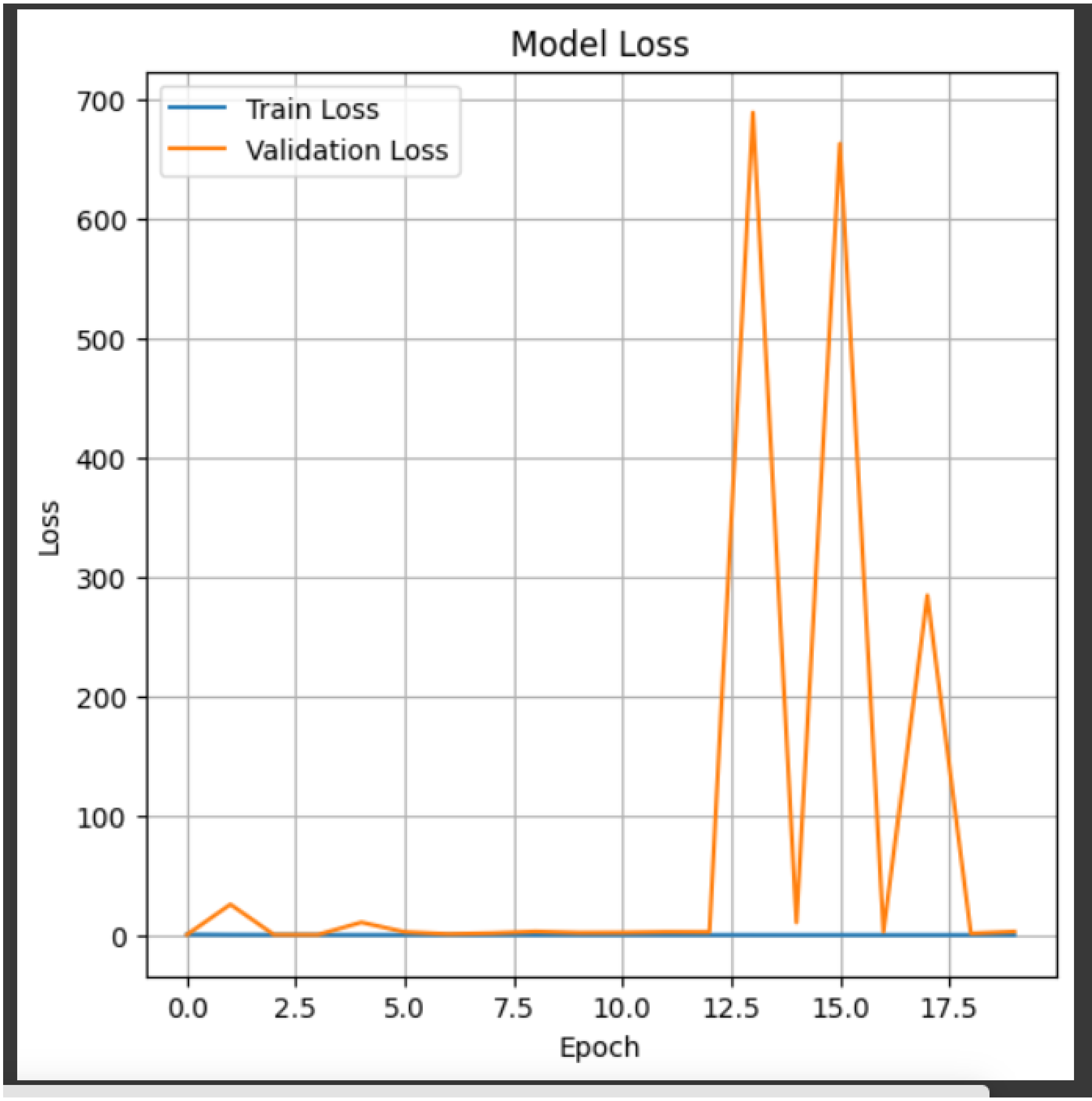
Second Case:

We used AlexNet .


Model Loss
Train Loss
Validation Loss
Loss
Epoch
0.60
0.55
0.50
0.45
0.40
0.35
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Model Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epoch
0.78
0.80
0.82
0.84
0.86
0.88
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

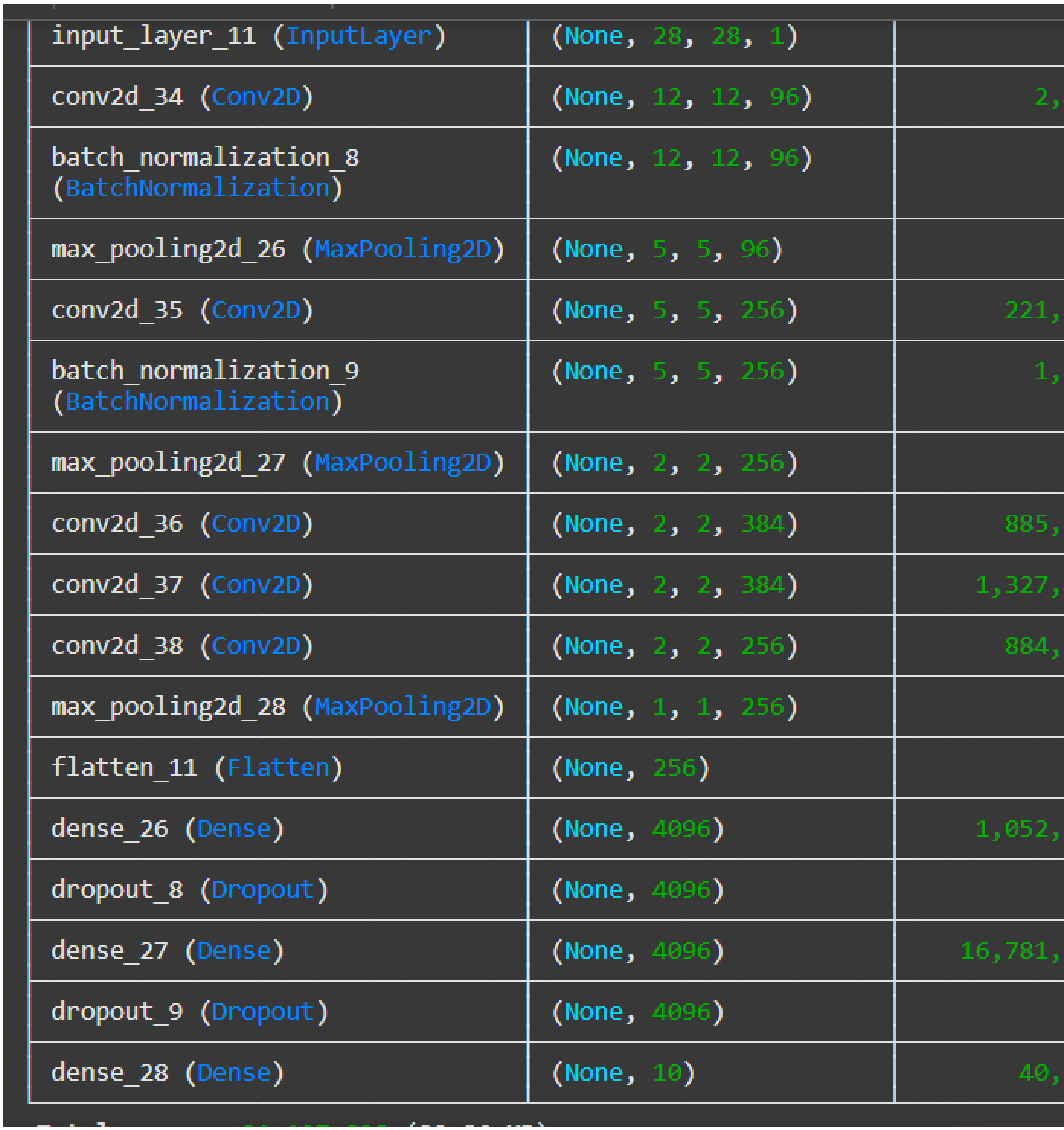

Model Accuracy
Accuracy
Epoch
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Train Accuracy
Validation Accuracy



Here we ran it two times ,one with 5 epochs, the other with 20 epochs

Either way ,it showed that the values do not stabilize at all

Here is the model



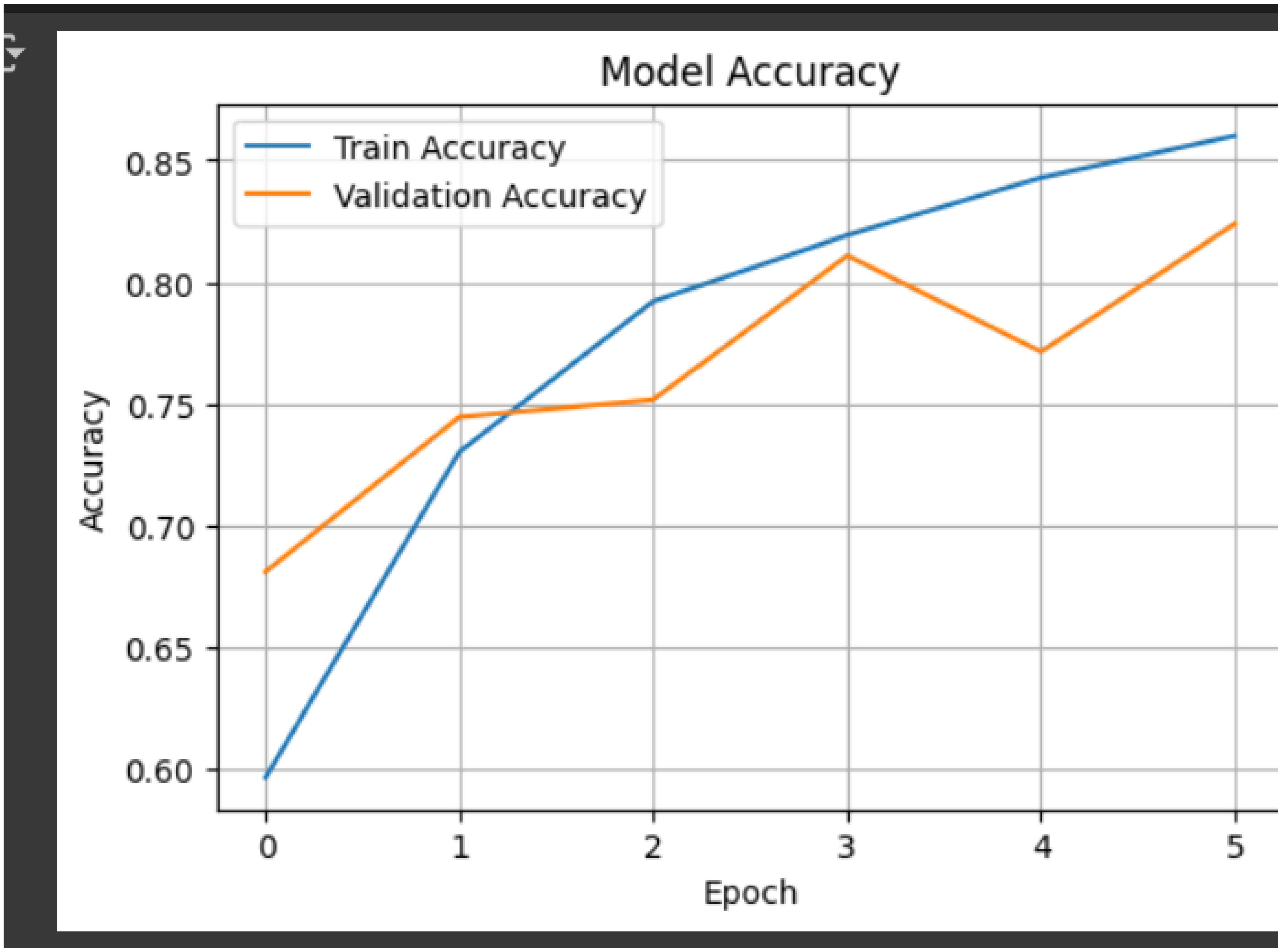
```
│ input_layer_11 (InputLayer)       │ (None, 28, 28, 1)        │
│ conv2d_34 (Conv2D)                │ (None, 12, 12, 96)       │          2,
│ batch_normalization_8             │ (None, 12, 12, 96)       │
│ (BatchNormalization)              │                          │
│ max_pooling2d_26 (MaxPooling2D)   │ (None, 5, 5, 96)         │
│ conv2d_35 (Conv2D)                │ (None, 5, 5, 256)        │        221,
│ batch_normalization_9             │ (None, 5, 5, 256)        │          1,
│ (BatchNormalization)              │                          │
│ max_pooling2d_27 (MaxPooling2D)   │ (None, 2, 2, 256)        │
│ conv2d_36 (Conv2D)                │ (None, 2, 2, 384)        │        885,
│ conv2d_37 (Conv2D)                │ (None, 2, 2, 384)        │      1,327,
│ conv2d_38 (Conv2D)                │ (None, 2, 2, 256)        │        884,
│ max_pooling2d_28 (MaxPooling2D)   │ (None, 1, 1, 256)        │
│ flatten_11 (Flatten)              │ (None, 256)              │
│ dense_26 (Dense)                  │ (None, 4096)             │      1,052,
│ dropout_8 (Dropout)               │ (None, 4096)             │
│ dense_27 (Dense)                  │ (None, 4096)             │     16,781,
│ dropout_9 (Dropout)               │ (None, 4096)             │
│ dense_28 (Dense)                  │ (None, 10)               │         40,
```

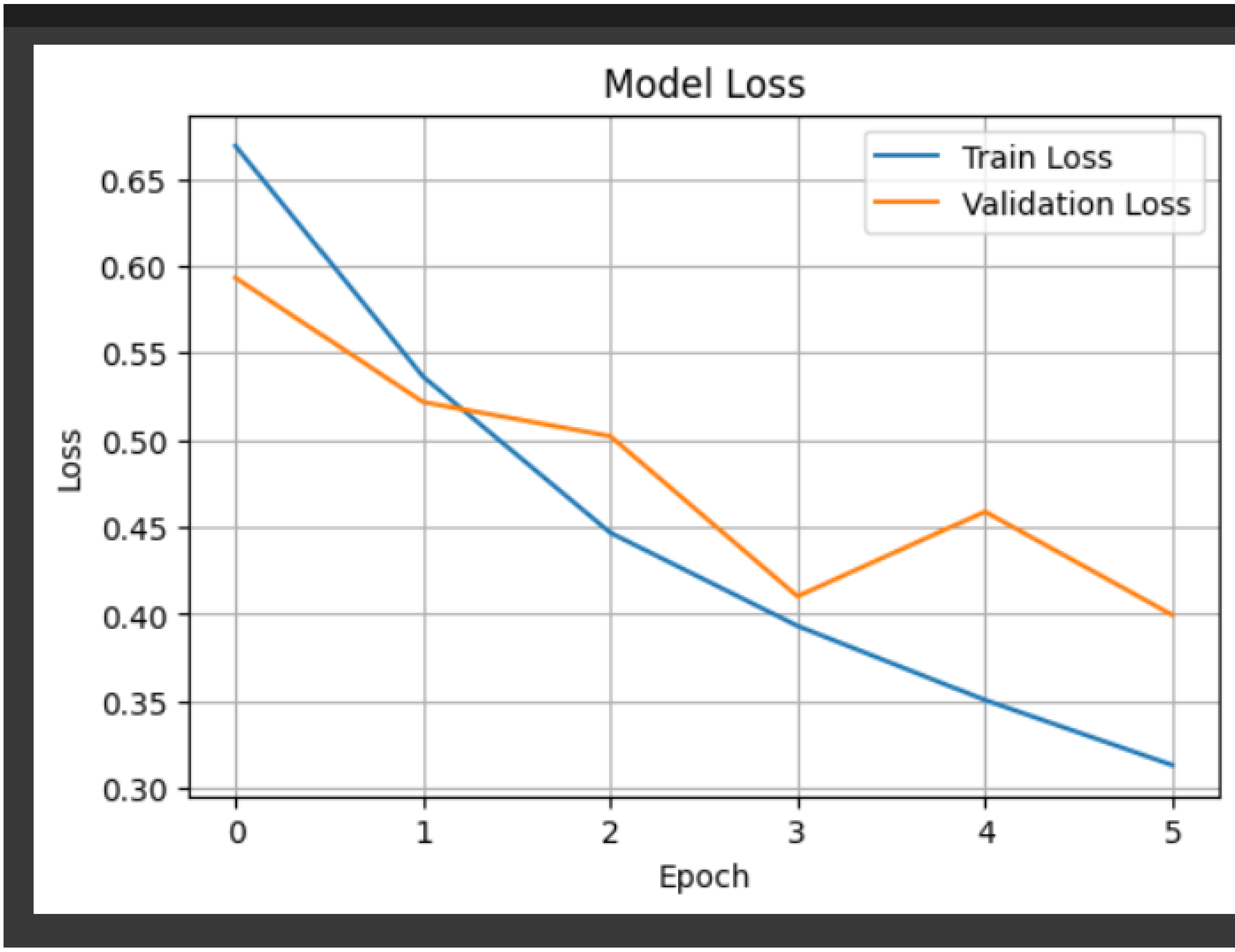
Class 2: Cats v Dogs

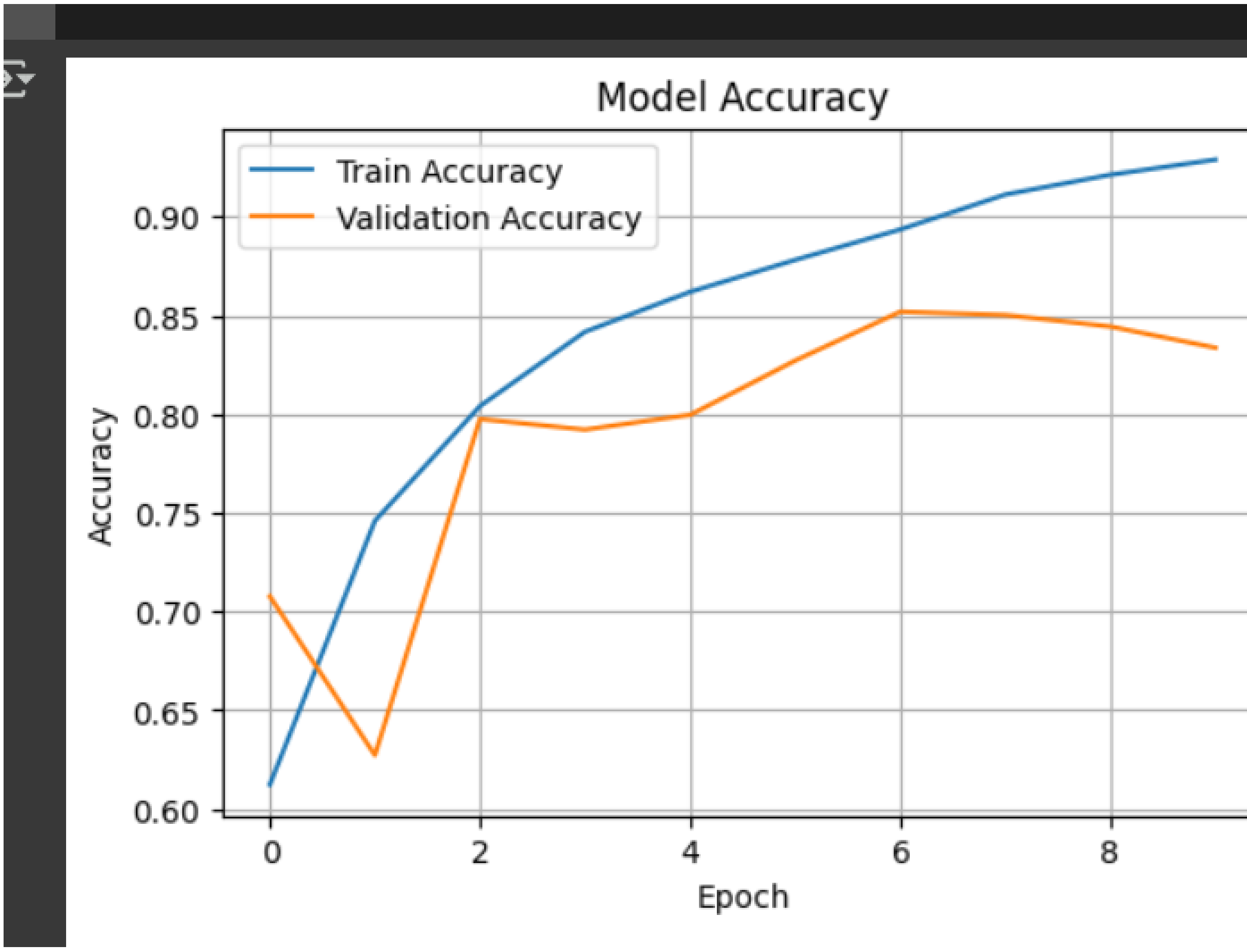
This case was pretty cool. I imagine Alex_Net won here because not only does it have to choose between only two subjects but also because Alex_Net is designed for image convolution

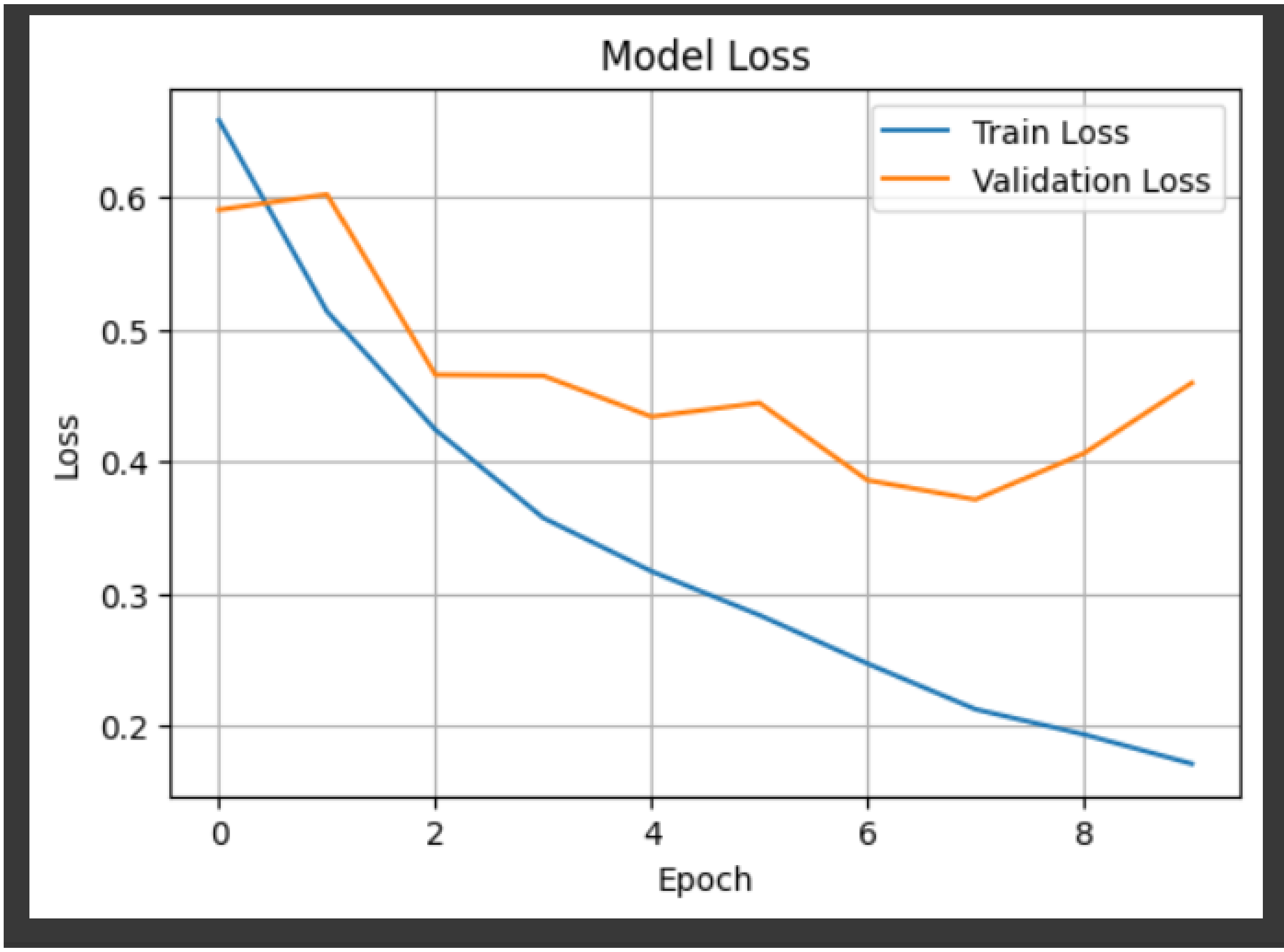
In the below graphs we see accuracy mostly increasing and loses decreasing, there is a time problem though as it takes more than 20ms to run each row

We ran it for 6 epochs and 10 epochs respectively

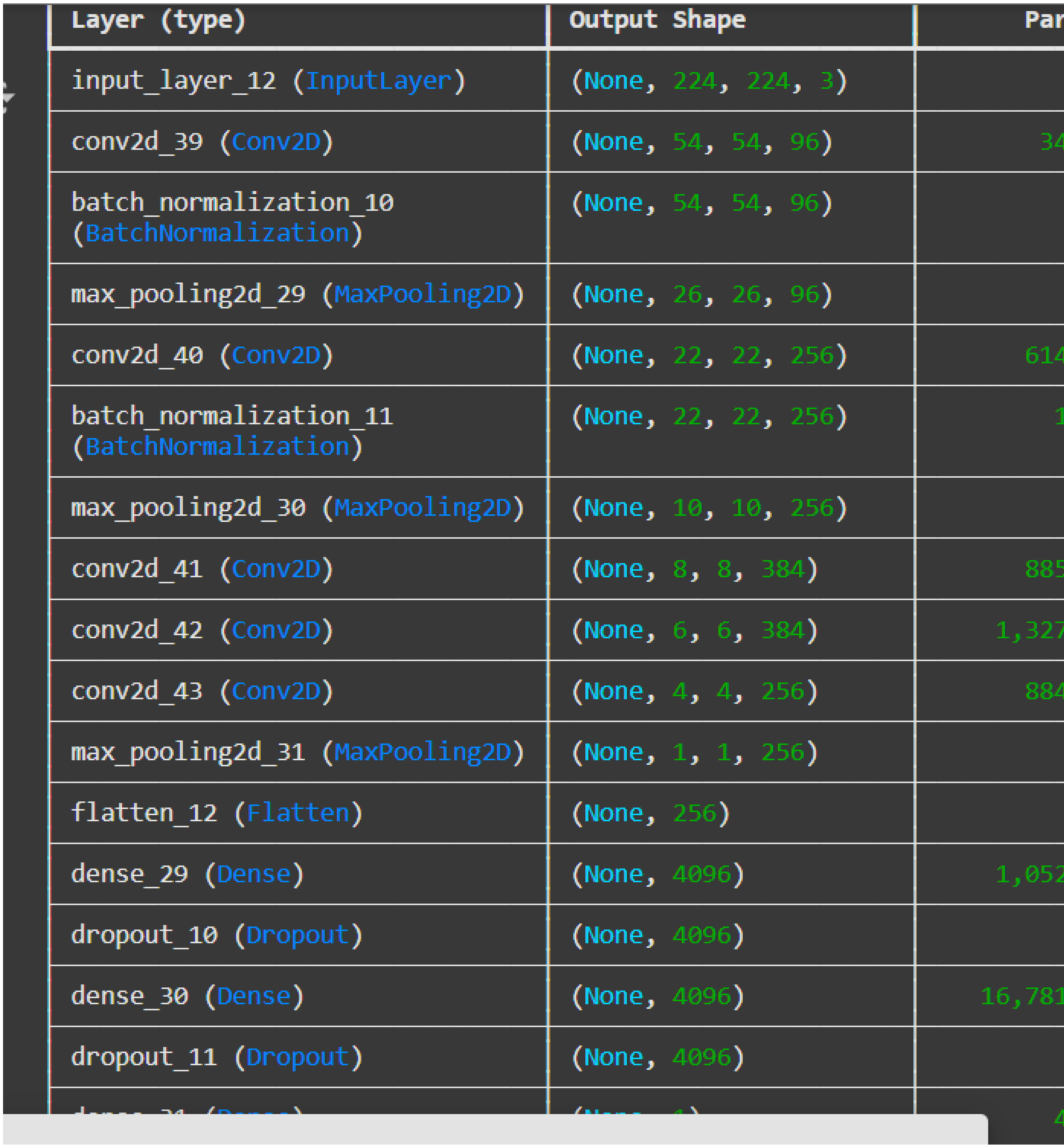

Model Accuracy
Train Accuracy
Validation Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
Accuracy
0
1
2
3
4
5
Epoch


Model Loss
Train Loss
Validation Loss
Loss
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0
1
2
3
4
5
Epoch


Model Accuracy
Train Accuracy
Validation Accuracy
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
2
4
6
8
Epoch



This is the model I used



```
Layer (type)                          Output Shape               Par
input_layer_12 (InputLayer)           (None, 224, 224, 3)
conv2d_39 (Conv2D)                    (None, 54, 54, 96)          34
batch_normalization_10                (None, 54, 54, 96)
(BatchNormalization)
max_pooling2d_29 (MaxPooling2D)       (None, 26, 26, 96)
conv2d_40 (Conv2D)                    (None, 22, 22, 256)        614
batch_normalization_11                (None, 22, 22, 256)          1
(BatchNormalization)
max_pooling2d_30 (MaxPooling2D)       (None, 10, 10, 256)
conv2d_41 (Conv2D)                    (None, 8, 8, 384)          885
conv2d_42 (Conv2D)                    (None, 6, 6, 384)        1,327
conv2d_43 (Conv2D)                    (None, 4, 4, 256)          884
max_pooling2d_31 (MaxPooling2D)       (None, 1, 1, 256)
flatten_12 (Flatten)                  (None, 256)
dense_29 (Dense)                      (None, 4096)             1,052
dropout_10 (Dropout)                  (None, 4096)
dense_30 (Dense)                      (None, 4096)            16,781
dropout_11 (Dropout)                  (None, 4096)
dense_31 (Dense)                      (None, 1)                    4
```

Training Error and Warning

A `ValueError` was encountered during training related to TensorFlow’s internal iterator mechanism. The traceback error message is provided below for reference. This may be due to an issue in the environment or data pipeline.

```
ValueError: Attr 'Toutput_types' of 'OptionalFromValue' Op passed list of length 0 less than minimum 1.
```

Why Fashion MNIST is "Easier" than Cats vs. Dogs

Despite having 10 classes compared to 2 for cats vs. dogs, Fashion MNIST is generally considered an "easier" classification task for machine learning models due to:

- **Image Simplicity:** Fashion MNIST images are low-resolution (28x28), grayscale, and highly standardized (centered items on plain backgrounds).
- **Reduced Variability:** Within each class (e.g., "T-shirt"), there's limited variation in pose, background, lighting, and specific sub-types.
- **Distinct Features:** The visual features distinguishing clothing items (e.g., the shape of a shoe vs. a coat) are relatively clear-cut and less ambiguous than distinguishing between different breeds or poses of real-world animals. In contrast, real-world datasets like "cats vs. dogs" feature high-resolution color images with diverse backgrounds, poses, lighting, and significant intra-class variation (different breeds, ages). This requires models to learn far more complex and abstract features.

Adapted AlexNet Architecture for 28x28 Images

To explore a more complex architecture, an AlexNet-like model was adapted for the 28x28 grayscale Fashion MNIST images. The original AlexNet was designed for larger (e.g., 224x224 or 227x227) 3-channel (RGB) images, so its initial convolutional and pooling layers had to be adjusted to prevent excessive downsampling of the smaller input.

Key adaptations include:

- **Input Shape:** Changed to (28, 28, 1) for grayscale images.
- **Initial Conv2D Layer:** kernel_size and strides were reduced (e.g., 5x5 kernel with stride 2) to maintain sufficient spatial information from the small input.
- **MaxPooling2D Layers:** pool_size and strides were adjusted to suit the smaller feature map sizes.
- **Output Layer:** Changed to Dense(10, activation='softmax') to correctly classify into 10 Fashion MNIST categories, replacing the original Dense(1, activation='sigmoid') which is typically for binary classification.
- **Batch Normalization:** Added after convolutional layers, which is a common practice in modern CNNs, often replacing AlexNet's original Local Response Normalization (LRN).

Error and Problems

A ValueError was encountered during training related to a mismatch between the input data shape and the model's expected input shape.

ValueError: Input 0 of layer "functional_8" is incompatible with the layer: expected shape=(None, 224, 224, 3), found shape=(32, 28, 28, 1).

There was also the problem that I was trying to run the convolution on dogs and cats but obviously it wouldn’t work because they are not greyscale images

Cause of the Error:

This error occurred because the model (which was an AlexNet-like architecture) was expecting input images of shape (None, 224, 224, 3) (meaning 224x224 pixels with 3 color channels), while the preprocessed Fashion MNIST data had a shape of (num_samples, 28, 28, 1) (28x28 pixels with 1 grayscale channel).

Resolution:

To resolve this, the Fashion MNIST images needed to be transformed to match the model's expected input:

1. **Resizing:** The 28x28 images were resized to 224x224 pixels using tf.image.resize.
2. **Channel Duplication:** The single grayscale channel was duplicated three times to create a 3-channel (RGB-like) image, even though the original content remained grayscale. This makes the data compatible with models expecting RGB input.

# Assuming x_train and x_test are already reshaped to (num_samples, 28, 28, 1) and normalized

# from previous preprocessing steps.

```
# 1. Resize images to 224x224
x_train_resized = tf.image.resize(x_train, (224,
224)).numpy()
x_test_resized = tf.image.resize(x_test, (224,
224)).numpy()

# 2. Convert grayscale (1 channel) to RGB (3
channels) by repeating the channel
x_train_rgb = np.repeat(x_train_resized, 3,
axis=-1)
x_test_rgb = np.repeat(x_test_resized, 3, axis=-
1)

print(f"x_train_rgb shape after resizing and
adding channels: {x_train_rgb.shape}")
print(f"x_test_rgb shape after resizing and
adding channels: {x_test_rgb.shape}")
```

These `x_train_rgb` and `x_test_rgb` would then be used for training the AlexNet-like model.

Conclusion

This report details the process of setting up and training CNN models on the Fashion MNIST dataset. It highlights the importance of correct data preprocessing, including normalization, reshaping, and splitting into training and validation sets.It also discuses the effects and advantages of Alex_Net and other convulations and tells us to when to use which. Furthermore, it addresses a common ValueError related to input shape mismatch and demonstrates how to adapt a more complex architecture like AlexNet for a smaller, grayscale dataset by adjusting its initial layers and input handling. The choice between a simpler CNN and an adapted AlexNet depends on the specific requirements and computational resources, with the simpler CNN often being more efficient for datasets like Fashion MNIST.