

midastask2part2-3

April 11, 2021

Author: Pushkar Patel

1 Task 2 Part 2

Imports

```
[1]: import shutil
from PIL import Image
import numpy as np
import scipy
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
%matplotlib inline
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Lambda
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import load_model
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
```

1.1 Preprocessing the image

Extracting the images

```
[ ]: shutil.unpack_archive('trainPart1.zip', 'input/part2')
```

Delete all other samples except images from 0-9

```
[ ]: for i in range(11, 63):
    shutil.rmtree(f'input/part2/train/Sample0{i}')
```

1.2 Generating the dataset for pre-training

As with part 1 of this task, I'm converting all the images to the dimension of the MNIST images by first resizing the largest dimension to 28 and then zer-padding them to make it a square. The background in the MNIST dataset have 0 value so, I'm inverting our dataset images to make it similar to MNIST

Some global variables

```
[2]: BATCH_SIZE = 64
      IMAGE_SIZE_BEFORE_PADDING = (21, 28)
      EPOCHS = 400
      IMAGE_SIZE = (28, 28)
```

Using ImageDataGenerator to generate the samples from the dataset

```
[ ]: train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2,
    ↪ preprocessing_function=lambda x: 1-x)
```

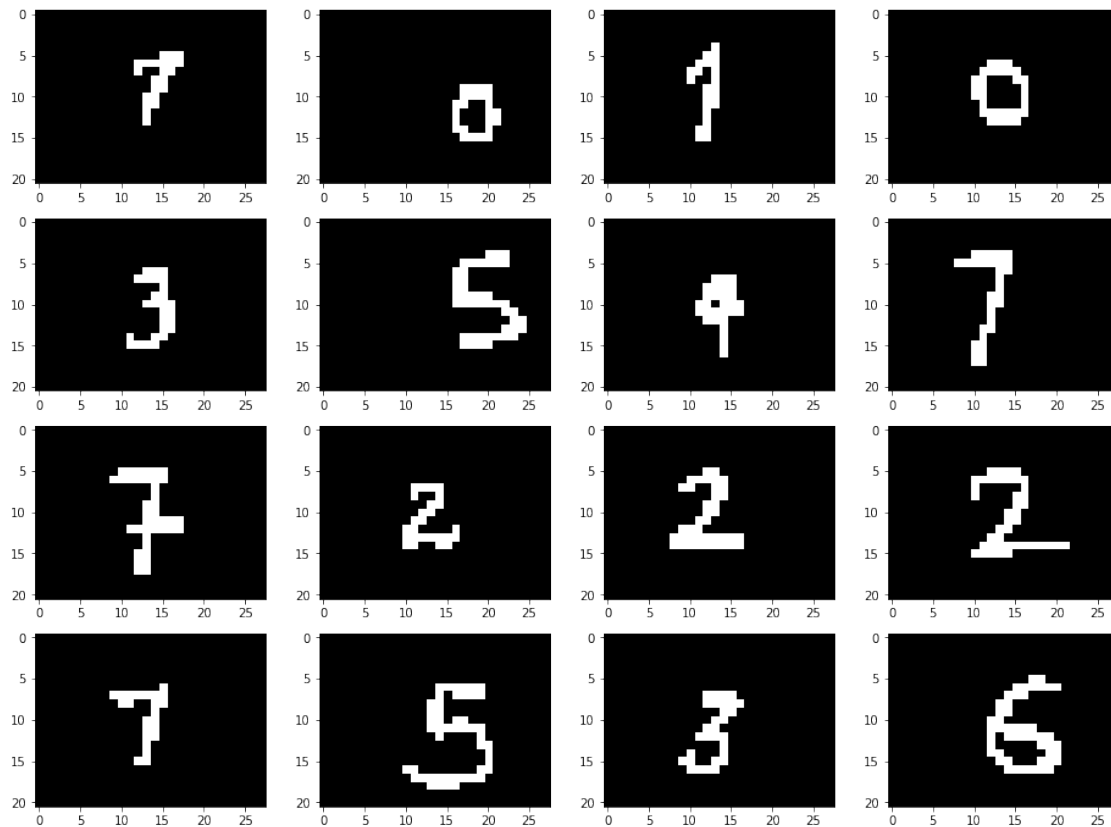
```
[ ]: train_generator = train_datagen.flow_from_directory(
    'input/part2/train',
    target_size=IMAGE_SIZE_BEFORE_PADDING,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    color_mode='grayscale',
    subset='training',
    seed=42,
    shuffle=True)

validation_generator = train_datagen.flow_from_directory(
    'input/part2/train',
    target_size=IMAGE_SIZE_BEFORE_PADDING,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    color_mode='grayscale',
    subset='validation',
    seed=42,
    shuffle=True)

X_train_batch0, y_train_batch0 = train_generator.next()
print(X_train_batch0.shape, y_train_batch0.shape)
print(y_train_batch0[0])
plt.figure(figsize=(16,12))
for i in range(1, 17):
    plt.subplot(4,4,i)
    imshow(tf.squeeze(X_train_batch0[i]), cmap='gray')
plt.show()
```

Found 320 images belonging to 10 classes.

Found 80 images belonging to 10 classes.
 (64, 21, 28, 1) (64, 10)
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]



1.3 Loading the MNIST Dataset

```
[3]: (x_train, y_train), (x_test, y_test) = mnist.load_data(path="mnist.npz")
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

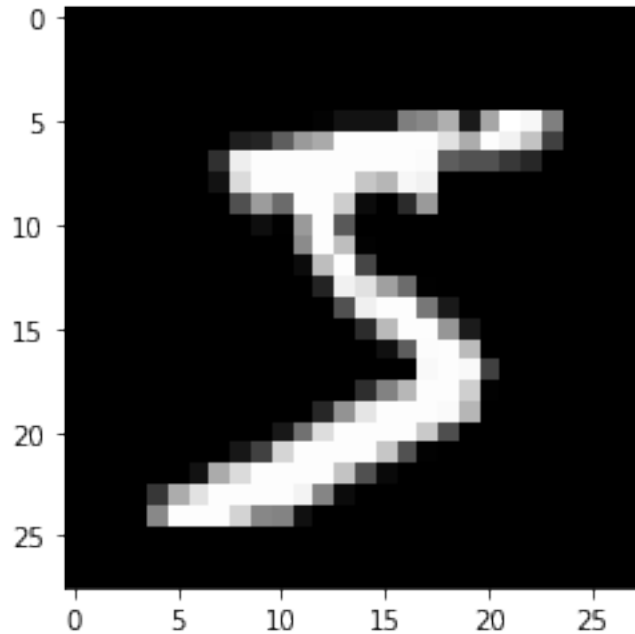
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[2], 1)
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1], x_test.shape[2], 1)

print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
imshow(tf.squeeze(x_train[0]), cmap='gray')
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>

```
11493376/11490434 [=====] - 0s 0us/step
(60000, 28, 28, 1) (60000, 10)
(10000, 28, 28, 1) (10000, 10)
```

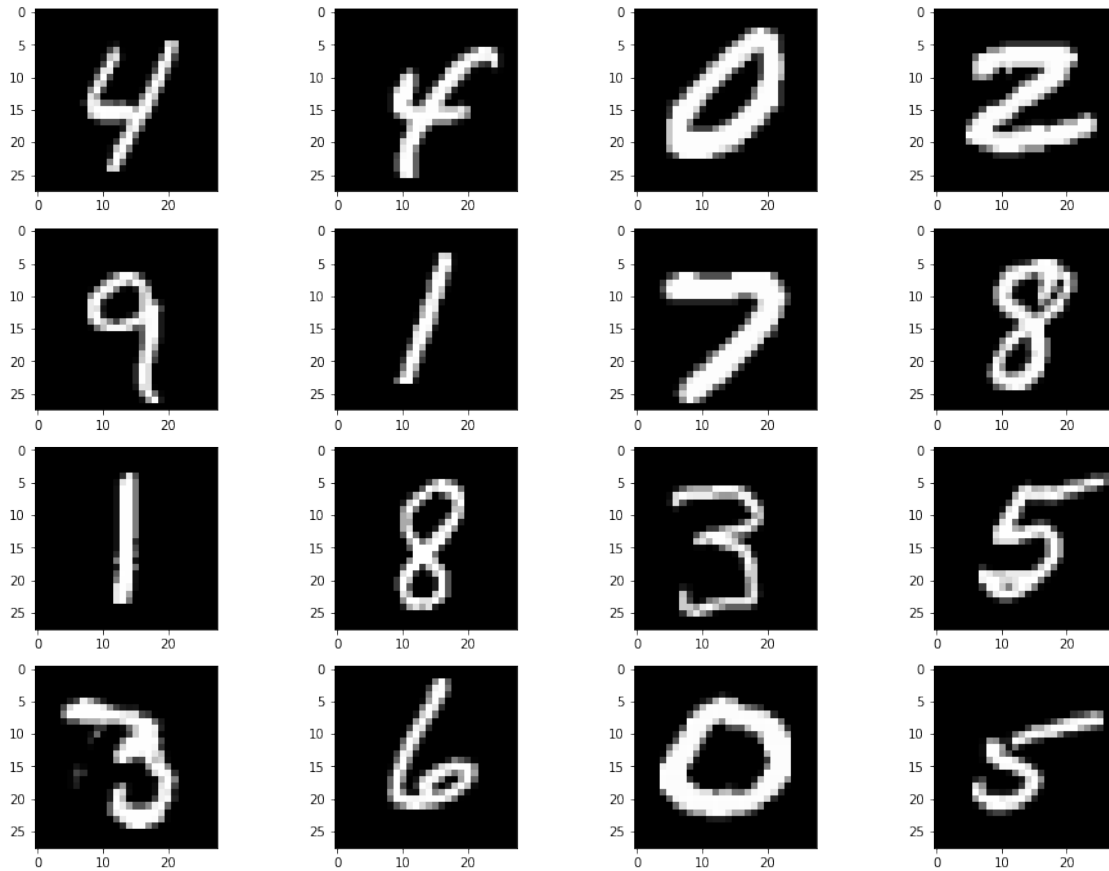
```
[3]: <matplotlib.image.AxesImage at 0x7ff7b01632d0>
```



```
[4]: mnist_datagen = ImageDataGenerator(rescale=1.0/255.0)

# prepare an iterators to scale images
mnist_train_gen = mnist_datagen.flow(x_train, y_train, batch_size=BATCH_SIZE)
mnist_test_gen = mnist_datagen.flow(x_test, y_test, batch_size=BATCH_SIZE)

mnist_x_train, _ = mnist_train_gen.next()
plt.figure(figsize=(16,12))
for i in range(1, 17):
    plt.subplot(4,4,i)
    imshow(tf.squeeze(mnist_x_train[i]), cmap='gray')
plt.show()
```



1.4 Building the Model

From my results from Part 1 of this task, Architecture from Kaggle and the original performed good. I'll use them both with Mish Activation and higher temperature for the Softmax and see how both of them perform.

```
[5]: # Early Stopping callback
early_stopping_callback = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    mode='min',
    patience=10,
    restore_best_weights=True,
    verbose=1)

early_stopping_callback2 = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    mode='min',
    patience=50,
    restore_best_weights=True,
    verbose=1)
```

```
[6]: # Softmax Temperature
temp = 5
```

```
[7]: # Mish Activation function
def mish(x):
    return tf.keras.layers.Lambda(lambda x: x*tf.tanh(tf.math.log(1+tf.
    ↪exp(x))))(x)
```

1.4.1 Modified LeNet Architecture

Pre-training on our dataset

```
[ ]: model1 = Sequential()

# Lambda Layer for adding Padding
model1.add(Lambda(lambda image: tf.image.resize_with_crop_or_pad(
    image, 28, 28), input_shape=(*IMAGE_SIZE_BEFORE_PADDING, 1)))

# 1st Convolution Layer
model1.add(Conv2D(6, input_shape=(*IMAGE_SIZE, 1),
    kernel_size=(5,5), padding='same', activation=mish))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size=(2,2), strides=2))

# 2nd Convolution Layer
model1.add(Conv2D(16, kernel_size=(5,5), activation=mish))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size=(2,2), strides=2))

# Passing to a Fully Connected Layer
model1.add(Flatten())

# 1st Fully Connected Layer
model1.add(Dense(120, activation=mish))
model1.add(BatchNormalization())
model1.add(Dropout(0.4))

# 2nd Fully Connected Layer
model1.add(Dense(84, activation=mish))
model1.add(BatchNormalization())
model1.add(Dropout(0.4))

# Output Layer
# Increasing the softmax temperature
model1.add(Lambda(lambda x: x / temp))
model1.add(Dense(10, activation='softmax'))

model1.summary()
```

```
model1.compile(loss='categorical_crossentropy', optimizer='adam',  
               metrics=['accuracy'])
```

Model: "sequential_9"

Layer (type)	Output Shape	Param #
lambda_18 (Lambda)	(None, 28, 28, 1)	0
conv2d_38 (Conv2D)	(None, 28, 28, 6)	156
batch_normalization_48 (Batch Normalization)	(None, 28, 28, 6)	24
max_pooling2d_10 (MaxPooling2D)	(None, 14, 14, 6)	0
conv2d_39 (Conv2D)	(None, 10, 10, 16)	2416
batch_normalization_49 (Batch Normalization)	(None, 10, 10, 16)	64
max_pooling2d_11 (MaxPooling2D)	(None, 5, 5, 16)	0
flatten_9 (Flatten)	(None, 400)	0
dense_19 (Dense)	(None, 120)	48120
batch_normalization_50 (Batch Normalization)	(None, 120)	480
dropout_22 (Dropout)	(None, 120)	0
dense_20 (Dense)	(None, 84)	10164
batch_normalization_51 (Batch Normalization)	(None, 84)	336
dropout_23 (Dropout)	(None, 84)	0
lambda_19 (Lambda)	(None, 84)	0
dense_21 (Dense)	(None, 10)	850
Total params: 62,610		
Trainable params: 62,158		
Non-trainable params: 452		

```
[ ]: checkpoint_filepath1 = 'part2_pretrained/checkpoint'  
     model_checkpoint_callback1 = tf.keras.callbacks.ModelCheckpoint(
```

```

        filepath=checkpoint_filepath1,
        save_weights_only=True,
        monitor='val_loss',
        mode='min',
        save_best_only=True)

history1 = model1.fit(
    train_generator,
    epochs=EPOCHS,
    validation_data=validation_generator,
    steps_per_epoch = train_generator.samples // BATCH_SIZE,
    validation_steps = validation_generator.samples // BATCH_SIZE,
    callbacks=[model_checkpoint_callback1, early_stopping_callback]
)

```

```

Epoch 1/400
5/5 [=====] - 7s 1s/step - loss: 2.3350 - accuracy:
0.1408 - val_loss: 2.3025 - val_accuracy: 0.1250
Epoch 2/400
5/5 [=====] - 5s 1s/step - loss: 2.1667 - accuracy:
0.1862 - val_loss: 2.3019 - val_accuracy: 0.1562
Epoch 3/400
5/5 [=====] - 5s 1s/step - loss: 2.0587 - accuracy:
0.3332 - val_loss: 2.2992 - val_accuracy: 0.2500
Epoch 4/400
5/5 [=====] - 5s 1s/step - loss: 1.9465 - accuracy:
0.4384 - val_loss: 2.3031 - val_accuracy: 0.0938
Epoch 5/400
5/5 [=====] - 5s 1s/step - loss: 1.8744 - accuracy:
0.5266 - val_loss: 2.2950 - val_accuracy: 0.1250
Epoch 6/400
5/5 [=====] - 5s 1s/step - loss: 1.7995 - accuracy:
0.5392 - val_loss: 2.2947 - val_accuracy: 0.0938
Epoch 7/400
5/5 [=====] - 5s 1s/step - loss: 1.7303 - accuracy:
0.6256 - val_loss: 2.2930 - val_accuracy: 0.0938
Epoch 8/400
5/5 [=====] - 5s 1s/step - loss: 1.6355 - accuracy:
0.6813 - val_loss: 2.2916 - val_accuracy: 0.0938
Epoch 9/400
5/5 [=====] - 5s 1s/step - loss: 1.5828 - accuracy:
0.7044 - val_loss: 2.2957 - val_accuracy: 0.0781
Epoch 10/400
5/5 [=====] - 5s 1s/step - loss: 1.5233 - accuracy:
0.7276 - val_loss: 2.2877 - val_accuracy: 0.1094
Epoch 11/400
5/5 [=====] - 5s 1s/step - loss: 1.4812 - accuracy:

```


0.7599 - val_loss: 2.2909 - val_accuracy: 0.0938
 Epoch 12/400
 5/5 [=====] - 5s 1s/step - loss: 1.3877 - accuracy:
 0.7908 - val_loss: 2.2846 - val_accuracy: 0.1094
 Epoch 13/400
 5/5 [=====] - 5s 1s/step - loss: 1.3261 - accuracy:
 0.7970 - val_loss: 2.2914 - val_accuracy: 0.0781
 Epoch 14/400
 5/5 [=====] - 5s 1s/step - loss: 1.2805 - accuracy:
 0.8424 - val_loss: 2.2885 - val_accuracy: 0.0938
 Epoch 15/400
 5/5 [=====] - 5s 1s/step - loss: 1.2517 - accuracy:
 0.8400 - val_loss: 2.2887 - val_accuracy: 0.1250
 Epoch 16/400
 5/5 [=====] - 5s 1s/step - loss: 1.1849 - accuracy:
 0.8377 - val_loss: 2.2985 - val_accuracy: 0.0781
 Epoch 17/400
 5/5 [=====] - 5s 1s/step - loss: 1.1716 - accuracy:
 0.8437 - val_loss: 2.3015 - val_accuracy: 0.0625
 Epoch 18/400
 5/5 [=====] - 5s 1s/step - loss: 1.1299 - accuracy:
 0.8556 - val_loss: 2.2902 - val_accuracy: 0.0938
 Epoch 19/400
 5/5 [=====] - 5s 1s/step - loss: 1.0983 - accuracy:
 0.8705 - val_loss: 2.2715 - val_accuracy: 0.1094
 Epoch 20/400
 5/5 [=====] - 5s 1s/step - loss: 1.0094 - accuracy:
 0.9096 - val_loss: 2.3084 - val_accuracy: 0.0625
 Epoch 21/400
 5/5 [=====] - 5s 1s/step - loss: 0.9620 - accuracy:
 0.9091 - val_loss: 2.2951 - val_accuracy: 0.0938
 Epoch 22/400
 5/5 [=====] - 5s 1s/step - loss: 0.8908 - accuracy:
 0.9303 - val_loss: 2.2915 - val_accuracy: 0.0938
 Epoch 23/400
 5/5 [=====] - 5s 1s/step - loss: 0.8682 - accuracy:
 0.9439 - val_loss: 2.2891 - val_accuracy: 0.0938
 Epoch 24/400
 5/5 [=====] - 5s 1s/step - loss: 0.8186 - accuracy:
 0.9275 - val_loss: 2.2630 - val_accuracy: 0.1094
 Epoch 25/400
 5/5 [=====] - 5s 1s/step - loss: 0.7741 - accuracy:
 0.9478 - val_loss: 2.2757 - val_accuracy: 0.0781
 Epoch 26/400
 5/5 [=====] - 5s 1s/step - loss: 0.7337 - accuracy:
 0.9551 - val_loss: 2.2845 - val_accuracy: 0.0938
 Epoch 27/400
 5/5 [=====] - 5s 1s/step - loss: 0.7269 - accuracy:

0.9504 - val_loss: 2.2639 - val_accuracy: 0.1094
 Epoch 28/400
 5/5 [=====] - 5s 1s/step - loss: 0.6759 - accuracy:
 0.9553 - val_loss: 2.2978 - val_accuracy: 0.0938
 Epoch 29/400
 5/5 [=====] - 5s 1s/step - loss: 0.6449 - accuracy:
 0.9685 - val_loss: 2.2644 - val_accuracy: 0.0938
 Epoch 30/400
 5/5 [=====] - 5s 1s/step - loss: 0.6143 - accuracy:
 0.9737 - val_loss: 2.2837 - val_accuracy: 0.1250
 Epoch 31/400
 5/5 [=====] - 5s 1s/step - loss: 0.5540 - accuracy:
 0.9898 - val_loss: 2.2901 - val_accuracy: 0.0938
 Epoch 32/400
 5/5 [=====] - 5s 1s/step - loss: 0.5415 - accuracy:
 0.9746 - val_loss: 2.2707 - val_accuracy: 0.1250
 Epoch 33/400
 5/5 [=====] - 5s 1s/step - loss: 0.5386 - accuracy:
 0.9858 - val_loss: 2.2405 - val_accuracy: 0.1562
 Epoch 34/400
 5/5 [=====] - 5s 1s/step - loss: 0.4734 - accuracy:
 0.9863 - val_loss: 2.2529 - val_accuracy: 0.1406
 Epoch 35/400
 5/5 [=====] - 5s 1s/step - loss: 0.5061 - accuracy:
 0.9815 - val_loss: 2.3207 - val_accuracy: 0.0938
 Epoch 36/400
 5/5 [=====] - 5s 1s/step - loss: 0.4500 - accuracy:
 0.9918 - val_loss: 2.2306 - val_accuracy: 0.1562
 Epoch 37/400
 5/5 [=====] - 5s 1s/step - loss: 0.4188 - accuracy:
 0.9874 - val_loss: 2.2606 - val_accuracy: 0.1250
 Epoch 38/400
 5/5 [=====] - 5s 1s/step - loss: 0.4009 - accuracy:
 0.9884 - val_loss: 2.2462 - val_accuracy: 0.1094
 Epoch 39/400
 5/5 [=====] - 5s 1s/step - loss: 0.3854 - accuracy:
 0.9832 - val_loss: 2.2524 - val_accuracy: 0.1250
 Epoch 40/400
 5/5 [=====] - 5s 1s/step - loss: 0.3878 - accuracy:
 0.9944 - val_loss: 2.2173 - val_accuracy: 0.1406
 Epoch 41/400
 5/5 [=====] - 5s 1s/step - loss: 0.3504 - accuracy:
 0.9990 - val_loss: 2.2385 - val_accuracy: 0.1094
 Epoch 42/400
 5/5 [=====] - 5s 1s/step - loss: 0.3115 - accuracy:
 0.9990 - val_loss: 2.2221 - val_accuracy: 0.1719
 Epoch 43/400
 5/5 [=====] - 5s 1s/step - loss: 0.3233 - accuracy:

```

1.0000 - val_loss: 2.2036 - val_accuracy: 0.1719
Epoch 44/400
5/5 [=====] - 5s 1s/step - loss: 0.2786 - accuracy:
1.0000 - val_loss: 2.2289 - val_accuracy: 0.1719
Epoch 45/400
5/5 [=====] - 5s 1s/step - loss: 0.2823 - accuracy:
0.9919 - val_loss: 2.2786 - val_accuracy: 0.1406
Epoch 46/400
5/5 [=====] - 5s 1s/step - loss: 0.2678 - accuracy:
1.0000 - val_loss: 2.1884 - val_accuracy: 0.2344
Epoch 47/400
5/5 [=====] - 5s 1s/step - loss: 0.2516 - accuracy:
0.9944 - val_loss: 2.1898 - val_accuracy: 0.2031
Epoch 48/400
5/5 [=====] - 5s 1s/step - loss: 0.2470 - accuracy:
1.0000 - val_loss: 2.1143 - val_accuracy: 0.2656
Epoch 49/400
5/5 [=====] - 5s 1s/step - loss: 0.2421 - accuracy:
1.0000 - val_loss: 2.1382 - val_accuracy: 0.2812
Epoch 50/400
5/5 [=====] - 5s 1s/step - loss: 0.2127 - accuracy:
0.9983 - val_loss: 2.1238 - val_accuracy: 0.2812
Epoch 51/400
5/5 [=====] - 5s 1s/step - loss: 0.2142 - accuracy:
1.0000 - val_loss: 2.1230 - val_accuracy: 0.2812
Epoch 52/400
5/5 [=====] - 5s 1s/step - loss: 0.2120 - accuracy:
1.0000 - val_loss: 2.1204 - val_accuracy: 0.2656
Epoch 53/400
5/5 [=====] - 5s 1s/step - loss: 0.2052 - accuracy:
0.9974 - val_loss: 2.0434 - val_accuracy: 0.2812
Epoch 54/400
5/5 [=====] - 5s 1s/step - loss: 0.1759 - accuracy:
1.0000 - val_loss: 2.1185 - val_accuracy: 0.2656
Epoch 55/400
5/5 [=====] - 5s 1s/step - loss: 0.1805 - accuracy:
1.0000 - val_loss: 2.1326 - val_accuracy: 0.2500
Epoch 56/400
5/5 [=====] - 5s 1s/step - loss: 0.1809 - accuracy:
1.0000 - val_loss: 2.1546 - val_accuracy: 0.2500
Epoch 57/400
5/5 [=====] - 5s 1s/step - loss: 0.1725 - accuracy:
1.0000 - val_loss: 2.0024 - val_accuracy: 0.3281
Epoch 58/400
5/5 [=====] - 5s 1s/step - loss: 0.1708 - accuracy:
0.9961 - val_loss: 2.0347 - val_accuracy: 0.3125
Epoch 59/400
5/5 [=====] - 5s 1s/step - loss: 0.1589 - accuracy:

```

1.0000 - val_loss: 1.9952 - val_accuracy: 0.3594
 Epoch 60/400
 5/5 [=====] - 5s 1s/step - loss: 0.1594 - accuracy:
 1.0000 - val_loss: 1.9768 - val_accuracy: 0.3438
 Epoch 61/400
 5/5 [=====] - 5s 1s/step - loss: 0.1380 - accuracy:
 1.0000 - val_loss: 1.9830 - val_accuracy: 0.3438
 Epoch 62/400
 5/5 [=====] - 5s 1s/step - loss: 0.1325 - accuracy:
 0.9974 - val_loss: 1.9374 - val_accuracy: 0.3438
 Epoch 63/400
 5/5 [=====] - 5s 1s/step - loss: 0.1380 - accuracy:
 1.0000 - val_loss: 1.9140 - val_accuracy: 0.3906
 Epoch 64/400
 5/5 [=====] - 5s 1s/step - loss: 0.1334 - accuracy:
 1.0000 - val_loss: 2.0197 - val_accuracy: 0.3438
 Epoch 65/400
 5/5 [=====] - 5s 1s/step - loss: 0.1328 - accuracy:
 1.0000 - val_loss: 1.8738 - val_accuracy: 0.4375
 Epoch 66/400
 5/5 [=====] - 5s 1s/step - loss: 0.1142 - accuracy:
 1.0000 - val_loss: 1.8837 - val_accuracy: 0.3594
 Epoch 67/400
 5/5 [=====] - 5s 1s/step - loss: 0.1263 - accuracy:
 1.0000 - val_loss: 1.8314 - val_accuracy: 0.4531
 Epoch 68/400
 5/5 [=====] - 5s 1s/step - loss: 0.1217 - accuracy:
 1.0000 - val_loss: 1.8422 - val_accuracy: 0.4219
 Epoch 69/400
 5/5 [=====] - 5s 1s/step - loss: 0.1003 - accuracy:
 1.0000 - val_loss: 1.8616 - val_accuracy: 0.4062
 Epoch 70/400
 5/5 [=====] - 5s 1s/step - loss: 0.0995 - accuracy:
 1.0000 - val_loss: 1.7578 - val_accuracy: 0.4844
 Epoch 71/400
 5/5 [=====] - 5s 1s/step - loss: 0.1037 - accuracy:
 1.0000 - val_loss: 1.8136 - val_accuracy: 0.4062
 Epoch 72/400
 5/5 [=====] - 5s 1s/step - loss: 0.1030 - accuracy:
 1.0000 - val_loss: 1.7199 - val_accuracy: 0.4375
 Epoch 73/400
 5/5 [=====] - 5s 1s/step - loss: 0.0939 - accuracy:
 1.0000 - val_loss: 1.7787 - val_accuracy: 0.4375
 Epoch 74/400
 5/5 [=====] - 5s 1s/step - loss: 0.0881 - accuracy:
 1.0000 - val_loss: 1.7015 - val_accuracy: 0.4375
 Epoch 75/400
 5/5 [=====] - 5s 1s/step - loss: 0.0795 - accuracy:

1.0000 - val_loss: 1.7746 - val_accuracy: 0.3750
Epoch 76/400
5/5 [=====] - 5s 1s/step - loss: 0.0826 - accuracy:
1.0000 - val_loss: 1.6884 - val_accuracy: 0.4219
Epoch 77/400
5/5 [=====] - 5s 1s/step - loss: 0.0821 - accuracy:
1.0000 - val_loss: 1.7874 - val_accuracy: 0.4219
Epoch 78/400
5/5 [=====] - 5s 1s/step - loss: 0.0844 - accuracy:
1.0000 - val_loss: 1.6324 - val_accuracy: 0.4688
Epoch 79/400
5/5 [=====] - 5s 1s/step - loss: 0.0875 - accuracy:
1.0000 - val_loss: 1.6147 - val_accuracy: 0.5156
Epoch 80/400
5/5 [=====] - 5s 1s/step - loss: 0.0747 - accuracy:
1.0000 - val_loss: 1.6822 - val_accuracy: 0.4844
Epoch 81/400
5/5 [=====] - 5s 1s/step - loss: 0.0756 - accuracy:
1.0000 - val_loss: 1.5401 - val_accuracy: 0.5156
Epoch 82/400
5/5 [=====] - 5s 1s/step - loss: 0.0680 - accuracy:
1.0000 - val_loss: 1.4624 - val_accuracy: 0.5000
Epoch 83/400
5/5 [=====] - 5s 1s/step - loss: 0.0690 - accuracy:
1.0000 - val_loss: 1.5235 - val_accuracy: 0.5156
Epoch 84/400
5/5 [=====] - 5s 1s/step - loss: 0.0722 - accuracy:
1.0000 - val_loss: 1.5455 - val_accuracy: 0.4844
Epoch 85/400
5/5 [=====] - 5s 1s/step - loss: 0.0737 - accuracy:
1.0000 - val_loss: 1.4869 - val_accuracy: 0.4844
Epoch 86/400
5/5 [=====] - 5s 1s/step - loss: 0.0702 - accuracy:
1.0000 - val_loss: 1.5180 - val_accuracy: 0.5312
Epoch 87/400
5/5 [=====] - 5s 1s/step - loss: 0.0675 - accuracy:
1.0000 - val_loss: 1.4989 - val_accuracy: 0.5000
Epoch 88/400
5/5 [=====] - 5s 1s/step - loss: 0.0680 - accuracy:
1.0000 - val_loss: 1.5496 - val_accuracy: 0.4688
Epoch 89/400
5/5 [=====] - 5s 1s/step - loss: 0.0653 - accuracy:
1.0000 - val_loss: 1.5138 - val_accuracy: 0.4688
Epoch 90/400
5/5 [=====] - 5s 1s/step - loss: 0.0614 - accuracy:
1.0000 - val_loss: 1.4448 - val_accuracy: 0.5156
Epoch 91/400
5/5 [=====] - 5s 1s/step - loss: 0.0684 - accuracy:

```

1.0000 - val_loss: 1.5538 - val_accuracy: 0.5000
Epoch 92/400
5/5 [=====] - 5s 1s/step - loss: 0.0565 - accuracy:
1.0000 - val_loss: 1.5061 - val_accuracy: 0.5156
Epoch 93/400
5/5 [=====] - 5s 1s/step - loss: 0.0626 - accuracy:
1.0000 - val_loss: 1.3431 - val_accuracy: 0.5625
Epoch 94/400
5/5 [=====] - 5s 1s/step - loss: 0.0518 - accuracy:
1.0000 - val_loss: 1.5286 - val_accuracy: 0.4844
Epoch 95/400
5/5 [=====] - 5s 1s/step - loss: 0.0546 - accuracy:
1.0000 - val_loss: 1.4344 - val_accuracy: 0.4844
Epoch 96/400
5/5 [=====] - 5s 1s/step - loss: 0.0594 - accuracy:
1.0000 - val_loss: 1.5791 - val_accuracy: 0.4844
Epoch 97/400
5/5 [=====] - 5s 1s/step - loss: 0.0468 - accuracy:
1.0000 - val_loss: 1.5325 - val_accuracy: 0.5156
Epoch 98/400
5/5 [=====] - 5s 1s/step - loss: 0.0552 - accuracy:
1.0000 - val_loss: 1.3494 - val_accuracy: 0.5469
Epoch 99/400
5/5 [=====] - 5s 1s/step - loss: 0.0459 - accuracy:
1.0000 - val_loss: 1.4215 - val_accuracy: 0.4844
Epoch 100/400
5/5 [=====] - 5s 1s/step - loss: 0.0448 - accuracy:
1.0000 - val_loss: 1.3978 - val_accuracy: 0.4844
Epoch 101/400
5/5 [=====] - 5s 1s/step - loss: 0.0435 - accuracy:
1.0000 - val_loss: 1.4117 - val_accuracy: 0.5000
Epoch 102/400
5/5 [=====] - 5s 1s/step - loss: 0.0421 - accuracy:
1.0000 - val_loss: 1.4643 - val_accuracy: 0.4688
Epoch 103/400
5/5 [=====] - 5s 1s/step - loss: 0.0439 - accuracy:
1.0000 - val_loss: 1.3935 - val_accuracy: 0.5312
Restoring model weights from the end of the best epoch.
Epoch 00103: early stopping

```

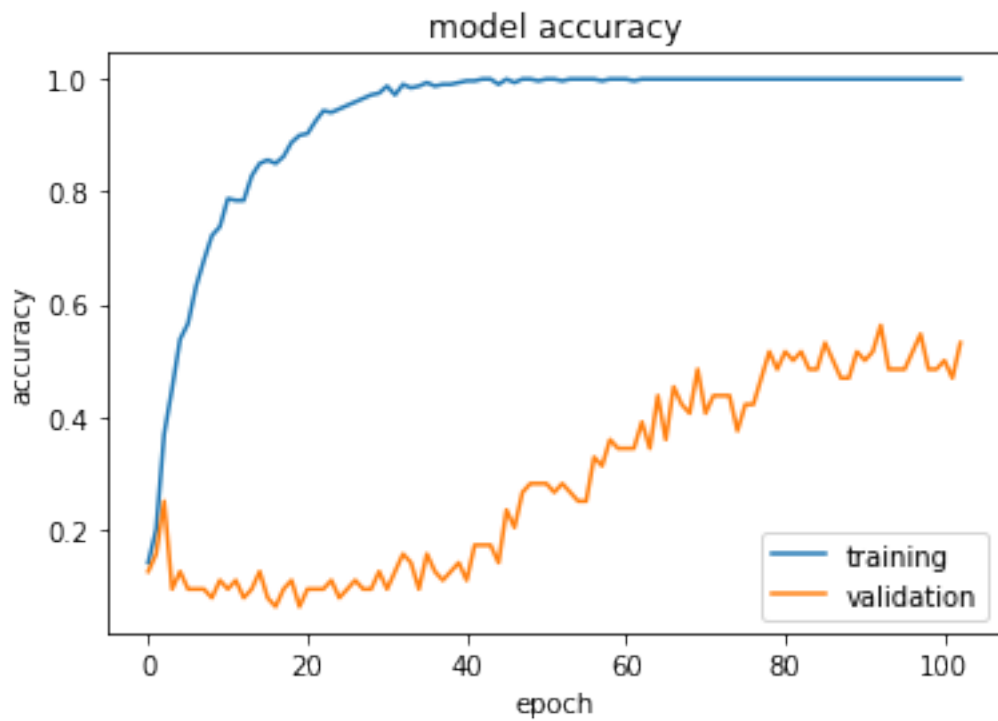
```

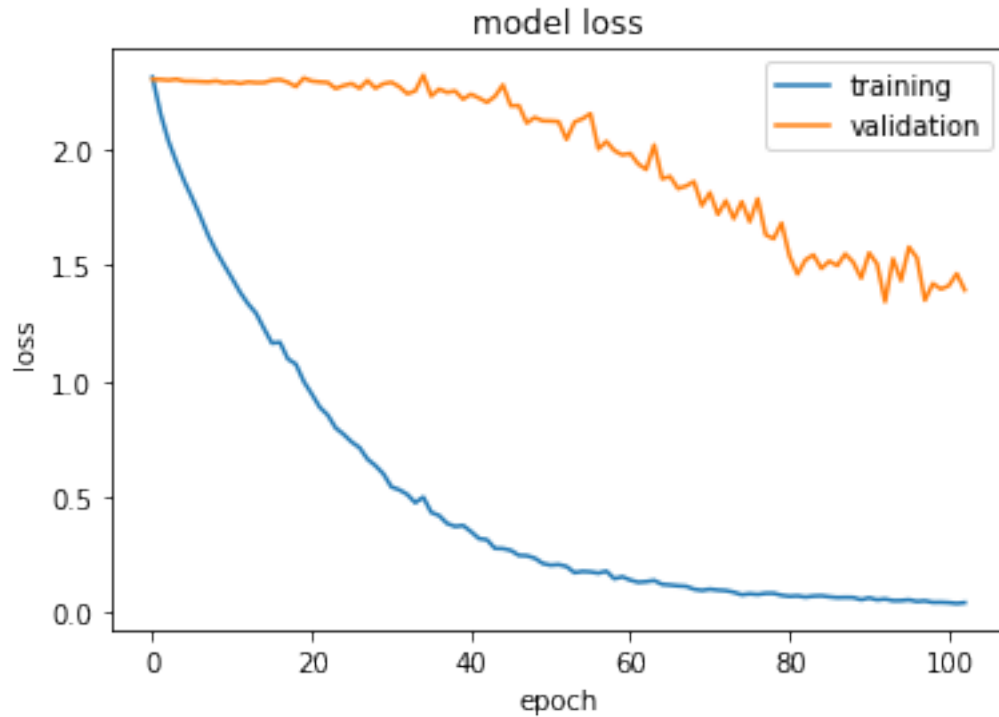
[ ]: plt.figure(1)
plt.plot(history1.history['accuracy'])
plt.plot(history1.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')

```

```
plt.show()

plt.figure(2)
plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```





This is the expected performance, but it actually gives good results after some more epochs. I think this is because now we have less samples to train with (only numbers).

Training on MNIST with pre-trained weights

```
[ ]: model1 = Sequential()

# Lambda Layer for adding Padding
model1.add(Lambda(lambda image: tf.image.resize_with_crop_or_pad(
    image, 28, 28), input_shape=(*IMAGE_SIZE_BEFORE_PADDING, 1)))

# 1st Convolution Layer
model1.add(Conv2D(6, input_shape=(*IMAGE_SIZE, 1),
    kernel_size=(5,5), padding='same', activation=mish))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size=(2,2), strides=2))

# 2nd Convolution Layer
model1.add(Conv2D(16, kernel_size=(5,5), activation=mish))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size=(2,2), strides=2))

# Passing to a Fully Connected Layer
model1.add(Flatten())
```



```

# 1st Fully Connected Layer
model1.add(Dense(120, activation=mish))
model1.add(BatchNormalization())
model1.add(Dropout(0.4))

# 2nd Fully Connected Layer
model1.add(Dense(84, activation=mish))
model1.add(BatchNormalization())
model1.add(Dropout(0.4))

# Output Layer
# Increasing the softmax temperature
model1.add(Lambda(lambda x: x / temp))
model1.add(Dense(10, activation='softmax'))

model1.summary()

model1.load_weights('part2_pretrained/checkpoint')

model1.compile(loss='categorical_crossentropy', optimizer=Adam(),
↳metrics=['accuracy'])

```

Model: "sequential_10"

Layer (type)	Output Shape	Param #
lambda_20 (Lambda)	(None, 28, 28, 1)	0
conv2d_40 (Conv2D)	(None, 28, 28, 6)	156
batch_normalization_52 (Batch Normalization)	(None, 28, 28, 6)	24
max_pooling2d_12 (MaxPooling2D)	(None, 14, 14, 6)	0
conv2d_41 (Conv2D)	(None, 10, 10, 16)	2416
batch_normalization_53 (Batch Normalization)	(None, 10, 10, 16)	64
max_pooling2d_13 (MaxPooling2D)	(None, 5, 5, 16)	0
flatten_10 (Flatten)	(None, 400)	0
dense_22 (Dense)	(None, 120)	48120
batch_normalization_54 (Batch Normalization)	(None, 120)	480

dropout_24 (Dropout)	(None, 120)	0

dense_23 (Dense)	(None, 84)	10164

batch_normalization_55 (Batch Normalization)	(None, 84)	336

dropout_25 (Dropout)	(None, 84)	0

lambda_21 (Lambda)	(None, 84)	0

dense_24 (Dense)	(None, 10)	850
=====		
Total params: 62,610		
Trainable params: 62,158		
Non-trainable params: 452		

```
[ ]: checkpoint_filepath1_after = 'part2_after_training/checkpoint'
model_checkpoint_callback1_after = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath1_after,
    save_weights_only=True,
    monitor='val_loss',
    mode='min',
    save_best_only=True)

history1_after = model1.fit(
    mnist_train_gen,
    epochs=EPOCHS,
    validation_data=mnist_test_gen,
    steps_per_epoch = len(x_train) // BATCH_SIZE,
    validation_steps = len(x_test) // BATCH_SIZE,
    callbacks=[model_checkpoint_callback1_after, early_stopping_callback]
)
```

Epoch 1/400

937/937 [=====] - 7s 7ms/step - loss: 0.3721 - accuracy: 0.8970 - val_loss: 0.0916 - val_accuracy: 0.9725

Epoch 2/400

937/937 [=====] - 6s 6ms/step - loss: 0.0984 - accuracy: 0.9727 - val_loss: 0.0605 - val_accuracy: 0.9818

Epoch 3/400

937/937 [=====] - 6s 6ms/step - loss: 0.0753 - accuracy: 0.9784 - val_loss: 0.0475 - val_accuracy: 0.9846

Epoch 4/400

937/937 [=====] - 6s 6ms/step - loss: 0.0618 - accuracy: 0.9826 - val_loss: 0.0421 - val_accuracy: 0.9865

Epoch 5/400

937/937 [=====] - 6s 6ms/step - loss: 0.0529 -

accuracy: 0.9849 - val_loss: 0.0447 - val_accuracy: 0.9853
 Epoch 6/400
 937/937 [=====] - 6s 6ms/step - loss: 0.0482 -
 accuracy: 0.9858 - val_loss: 0.0472 - val_accuracy: 0.9853
 Epoch 7/400
 937/937 [=====] - 6s 6ms/step - loss: 0.0456 -
 accuracy: 0.9857 - val_loss: 0.0505 - val_accuracy: 0.9848
 Epoch 8/400
 937/937 [=====] - 6s 6ms/step - loss: 0.0393 -
 accuracy: 0.9880 - val_loss: 0.0418 - val_accuracy: 0.9872
 Epoch 9/400
 937/937 [=====] - 6s 6ms/step - loss: 0.0423 -
 accuracy: 0.9871 - val_loss: 0.0398 - val_accuracy: 0.9878
 Epoch 10/400
 937/937 [=====] - 6s 7ms/step - loss: 0.0358 -
 accuracy: 0.9893 - val_loss: 0.0430 - val_accuracy: 0.9869
 Epoch 11/400
 937/937 [=====] - 6s 6ms/step - loss: 0.0328 -
 accuracy: 0.9899 - val_loss: 0.0410 - val_accuracy: 0.9882
 Epoch 12/400
 937/937 [=====] - 6s 6ms/step - loss: 0.0315 -
 accuracy: 0.9904 - val_loss: 0.0527 - val_accuracy: 0.9859
 Epoch 13/400
 937/937 [=====] - 6s 6ms/step - loss: 0.0307 -
 accuracy: 0.9914 - val_loss: 0.0391 - val_accuracy: 0.9891
 Epoch 14/400
 937/937 [=====] - 6s 7ms/step - loss: 0.0307 -
 accuracy: 0.9911 - val_loss: 0.0442 - val_accuracy: 0.9885
 Epoch 15/400
 937/937 [=====] - 6s 6ms/step - loss: 0.0287 -
 accuracy: 0.9913 - val_loss: 0.0378 - val_accuracy: 0.9902
 Epoch 16/400
 937/937 [=====] - 6s 6ms/step - loss: 0.0262 -
 accuracy: 0.9919 - val_loss: 0.0438 - val_accuracy: 0.9890
 Epoch 17/400
 937/937 [=====] - 6s 6ms/step - loss: 0.0259 -
 accuracy: 0.9921 - val_loss: 0.0417 - val_accuracy: 0.9875
 Epoch 18/400
 937/937 [=====] - 6s 6ms/step - loss: 0.0262 -
 accuracy: 0.9920 - val_loss: 0.0394 - val_accuracy: 0.9886
 Epoch 19/400
 937/937 [=====] - 6s 6ms/step - loss: 0.0235 -
 accuracy: 0.9925 - val_loss: 0.0413 - val_accuracy: 0.9882
 Epoch 20/400
 937/937 [=====] - 6s 6ms/step - loss: 0.0195 -
 accuracy: 0.9937 - val_loss: 0.0438 - val_accuracy: 0.9881
 Epoch 21/400
 937/937 [=====] - 6s 6ms/step - loss: 0.0215 -

```

accuracy: 0.9928 - val_loss: 0.0491 - val_accuracy: 0.9867
Epoch 22/400
937/937 [=====] - 6s 6ms/step - loss: 0.0214 -
accuracy: 0.9935 - val_loss: 0.0377 - val_accuracy: 0.9893
Epoch 23/400
937/937 [=====] - 6s 6ms/step - loss: 0.0227 -
accuracy: 0.9929 - val_loss: 0.0440 - val_accuracy: 0.9882
Epoch 24/400
937/937 [=====] - 6s 6ms/step - loss: 0.0218 -
accuracy: 0.9932 - val_loss: 0.0409 - val_accuracy: 0.9892
Epoch 25/400
937/937 [=====] - 6s 6ms/step - loss: 0.0213 -
accuracy: 0.9934 - val_loss: 0.0478 - val_accuracy: 0.9886
Epoch 26/400
937/937 [=====] - 6s 6ms/step - loss: 0.0180 -
accuracy: 0.9942 - val_loss: 0.0406 - val_accuracy: 0.9892
Epoch 27/400
937/937 [=====] - 6s 6ms/step - loss: 0.0147 -
accuracy: 0.9953 - val_loss: 0.0414 - val_accuracy: 0.9900
Epoch 28/400
937/937 [=====] - 7s 7ms/step - loss: 0.0163 -
accuracy: 0.9948 - val_loss: 0.0402 - val_accuracy: 0.9898
Epoch 29/400
937/937 [=====] - 6s 7ms/step - loss: 0.0172 -
accuracy: 0.9946 - val_loss: 0.0434 - val_accuracy: 0.9896
Epoch 30/400
937/937 [=====] - 6s 6ms/step - loss: 0.0183 -
accuracy: 0.9941 - val_loss: 0.0438 - val_accuracy: 0.9897
Epoch 31/400
937/937 [=====] - 6s 6ms/step - loss: 0.0136 -
accuracy: 0.9958 - val_loss: 0.0493 - val_accuracy: 0.9891
Epoch 32/400
937/937 [=====] - 6s 6ms/step - loss: 0.0155 -
accuracy: 0.9948 - val_loss: 0.0460 - val_accuracy: 0.9895
Restoring model weights from the end of the best epoch.
Epoch 00032: early stopping

```

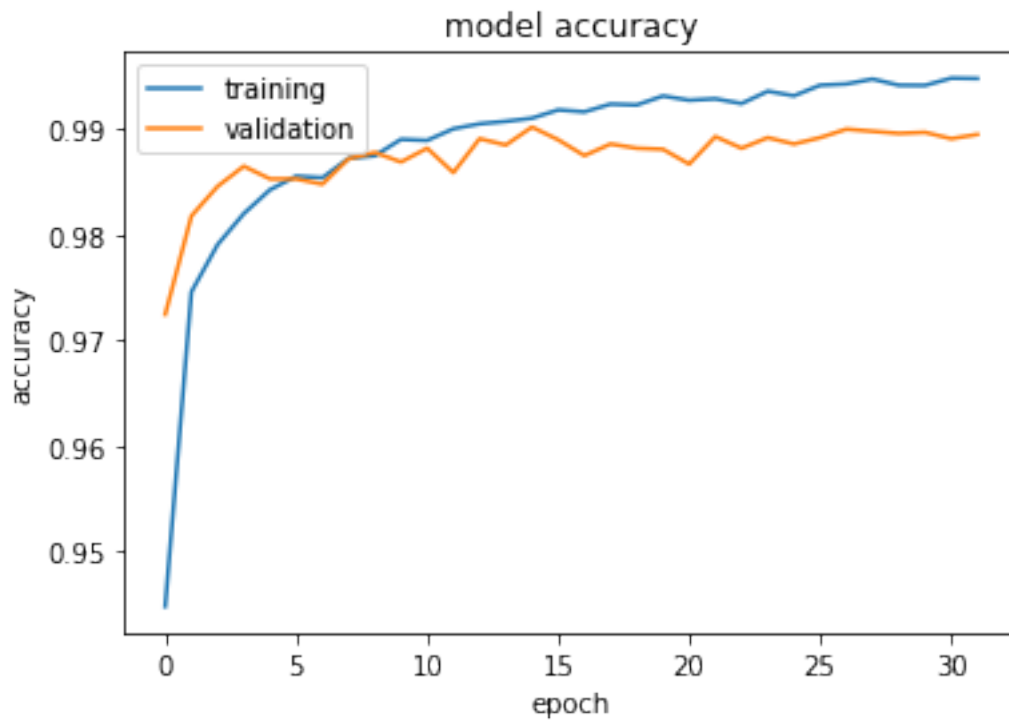
```

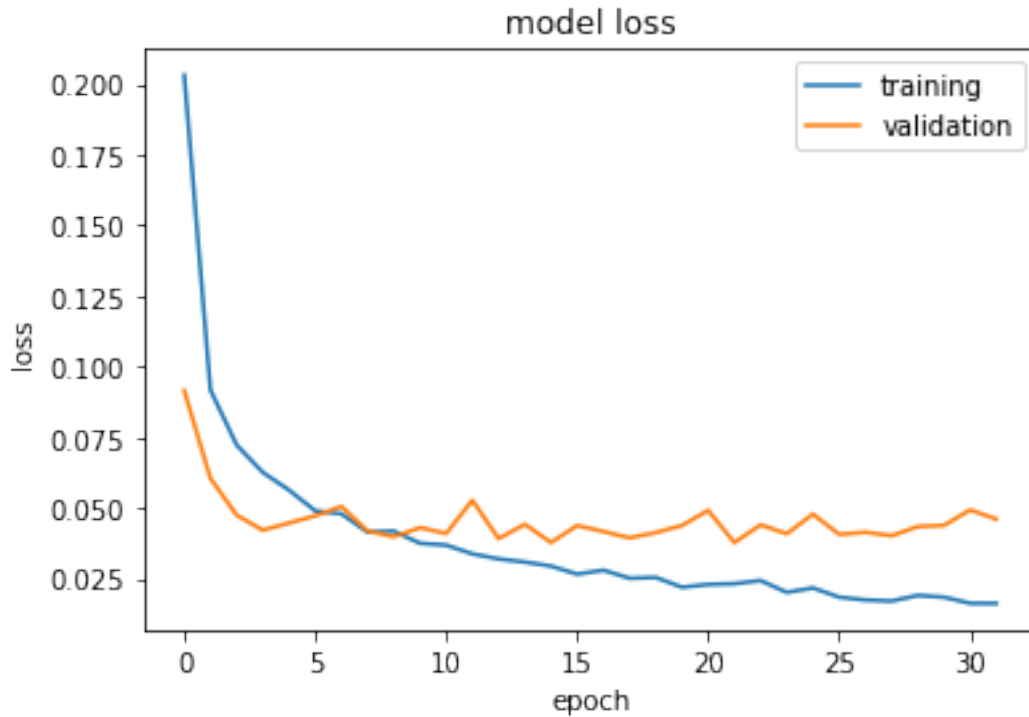
[ ]: plt.figure(1)
plt.plot(history1_after.history['accuracy'])
plt.plot(history1_after.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()

plt.figure(2)

```

```
plt.plot(history1_after.history['loss'])
plt.plot(history1_after.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```





This performs really good on the MNIST dataset and converges fast.

Untrained network with random initializations

```
[ ]: model1 = Sequential()

# Lambda Layer for adding Padding
model1.add(Lambda(lambda image: tf.image.resize_with_crop_or_pad(
    image, 28, 28), input_shape=(*IMAGE_SIZE_BEFORE_PADDING, 1)))

# 1st Convolution Layer
model1.add(Conv2D(6, input_shape=(*IMAGE_SIZE, 1),
    kernel_size=(5,5), padding='same', activation=mish))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size=(2,2), strides=2))

# 2nd Convolution Layer
model1.add(Conv2D(16, kernel_size=(5,5), activation=mish))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size=(2,2), strides=2))

# Passing to a Fully Connected Layer
model1.add(Flatten())
```

```

# 1st Fully Connected Layer
model1.add(Dense(120, activation=mish))
model1.add(BatchNormalization())
model1.add(Dropout(0.4))

# 2nd Fully Connected Layer
model1.add(Dense(84, activation=mish))
model1.add(BatchNormalization())
model1.add(Dropout(0.4))

# Output Layer
# Increasing the softmax temperature
model1.add(Lambda(lambda x: x / temp))
model1.add(Dense(10, activation='softmax'))

model1.summary()

model1.compile(loss='categorical_crossentropy', optimizer=Adam(),
               metrics=['accuracy'])

```

Model: "sequential_12"

Layer (type)	Output Shape	Param #
lambda_24 (Lambda)	(None, 28, 28, 1)	0
conv2d_44 (Conv2D)	(None, 28, 28, 6)	156
batch_normalization_60 (Batch Normalization)	(None, 28, 28, 6)	24
max_pooling2d_16 (MaxPooling2D)	(None, 14, 14, 6)	0
conv2d_45 (Conv2D)	(None, 10, 10, 16)	2416
batch_normalization_61 (Batch Normalization)	(None, 10, 10, 16)	64
max_pooling2d_17 (MaxPooling2D)	(None, 5, 5, 16)	0
flatten_12 (Flatten)	(None, 400)	0
dense_28 (Dense)	(None, 120)	48120
batch_normalization_62 (Batch Normalization)	(None, 120)	480
dropout_28 (Dropout)	(None, 120)	0
dense_29 (Dense)	(None, 84)	10164

```

-----
batch_normalization_63 (Batch Normalization) (None, 84) 336
-----
dropout_29 (Dropout) (None, 84) 0
-----
lambda_25 (Lambda) (None, 84) 0
-----
dense_30 (Dense) (None, 10) 850
=====
Total params: 62,610
Trainable params: 62,158
Non-trainable params: 452
-----

```

```

[ ]: checkpoint_filepath1_un = 'part2_untrained/checkpoint'
model_checkpoint_callback1_un = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath1_un,
    save_weights_only=True,
    monitor='val_loss',
    mode='min',
    save_best_only=True)

history1_un = model1.fit(
    mnist_train_gen,
    epochs=EPOCHS,
    validation_data=mnist_test_gen,
    steps_per_epoch = len(x_train) // BATCH_SIZE,
    validation_steps = len(x_test) // BATCH_SIZE,
    callbacks=[model_checkpoint_callback1_un, early_stopping_callback]
)

```

```

Epoch 1/400
937/937 [=====] - 7s 7ms/step - loss: 0.7657 -
accuracy: 0.8426 - val_loss: 0.1065 - val_accuracy: 0.9671
Epoch 2/400
937/937 [=====] - 6s 7ms/step - loss: 0.1026 -
accuracy: 0.9714 - val_loss: 0.0564 - val_accuracy: 0.9821
Epoch 3/400
937/937 [=====] - 6s 6ms/step - loss: 0.0768 -
accuracy: 0.9788 - val_loss: 0.0495 - val_accuracy: 0.9848
Epoch 4/400
937/937 [=====] - 6s 6ms/step - loss: 0.0648 -
accuracy: 0.9809 - val_loss: 0.0486 - val_accuracy: 0.9849
Epoch 5/400
937/937 [=====] - 6s 6ms/step - loss: 0.0569 -
accuracy: 0.9834 - val_loss: 0.0466 - val_accuracy: 0.9854
Epoch 6/400
937/937 [=====] - 6s 6ms/step - loss: 0.0544 -

```



```

accuracy: 0.9834 - val_loss: 0.0490 - val_accuracy: 0.9854
Epoch 7/400
937/937 [=====] - 6s 6ms/step - loss: 0.0533 -
accuracy: 0.9844 - val_loss: 0.0476 - val_accuracy: 0.9858
Epoch 8/400
937/937 [=====] - 6s 6ms/step - loss: 0.0425 -
accuracy: 0.9870 - val_loss: 0.0424 - val_accuracy: 0.9873
Epoch 9/400
937/937 [=====] - 6s 6ms/step - loss: 0.0412 -
accuracy: 0.9877 - val_loss: 0.0420 - val_accuracy: 0.9868
Epoch 10/400
937/937 [=====] - 6s 6ms/step - loss: 0.0429 -
accuracy: 0.9868 - val_loss: 0.0527 - val_accuracy: 0.9843
Epoch 11/400
937/937 [=====] - 6s 6ms/step - loss: 0.0374 -
accuracy: 0.9888 - val_loss: 0.0470 - val_accuracy: 0.9860
Epoch 12/400
937/937 [=====] - 6s 6ms/step - loss: 0.0326 -
accuracy: 0.9896 - val_loss: 0.0416 - val_accuracy: 0.9886
Epoch 13/400
937/937 [=====] - 6s 6ms/step - loss: 0.0355 -
accuracy: 0.9896 - val_loss: 0.0434 - val_accuracy: 0.9874
Epoch 14/400
937/937 [=====] - 6s 6ms/step - loss: 0.0311 -
accuracy: 0.9908 - val_loss: 0.0423 - val_accuracy: 0.9881
Epoch 15/400
937/937 [=====] - 6s 6ms/step - loss: 0.0304 -
accuracy: 0.9904 - val_loss: 0.0405 - val_accuracy: 0.9891
Epoch 16/400
937/937 [=====] - 6s 6ms/step - loss: 0.0305 -
accuracy: 0.9909 - val_loss: 0.0430 - val_accuracy: 0.9879
Epoch 17/400
937/937 [=====] - 6s 6ms/step - loss: 0.0275 -
accuracy: 0.9913 - val_loss: 0.0372 - val_accuracy: 0.9886
Epoch 18/400
937/937 [=====] - 6s 6ms/step - loss: 0.0256 -
accuracy: 0.9923 - val_loss: 0.0465 - val_accuracy: 0.9883
Epoch 19/400
937/937 [=====] - 6s 6ms/step - loss: 0.0260 -
accuracy: 0.9924 - val_loss: 0.0391 - val_accuracy: 0.9893
Epoch 20/400
937/937 [=====] - 6s 6ms/step - loss: 0.0237 -
accuracy: 0.9926 - val_loss: 0.0406 - val_accuracy: 0.9881
Epoch 21/400
937/937 [=====] - 6s 6ms/step - loss: 0.0234 -
accuracy: 0.9929 - val_loss: 0.0433 - val_accuracy: 0.9888
Epoch 22/400
937/937 [=====] - 6s 6ms/step - loss: 0.0212 -

```

```

accuracy: 0.9936 - val_loss: 0.0387 - val_accuracy: 0.9894
Epoch 23/400
937/937 [=====] - 6s 6ms/step - loss: 0.0241 -
accuracy: 0.9925 - val_loss: 0.0357 - val_accuracy: 0.9899
Epoch 24/400
937/937 [=====] - 6s 6ms/step - loss: 0.0207 -
accuracy: 0.9934 - val_loss: 0.0395 - val_accuracy: 0.9895
Epoch 25/400
937/937 [=====] - 6s 6ms/step - loss: 0.0206 -
accuracy: 0.9930 - val_loss: 0.0445 - val_accuracy: 0.9877
Epoch 26/400
937/937 [=====] - 6s 6ms/step - loss: 0.0213 -
accuracy: 0.9934 - val_loss: 0.0408 - val_accuracy: 0.9881
Epoch 27/400
937/937 [=====] - 6s 7ms/step - loss: 0.0205 -
accuracy: 0.9937 - val_loss: 0.0488 - val_accuracy: 0.9877
Epoch 28/400
937/937 [=====] - 6s 6ms/step - loss: 0.0195 -
accuracy: 0.9942 - val_loss: 0.0423 - val_accuracy: 0.9889
Epoch 29/400
937/937 [=====] - 6s 6ms/step - loss: 0.0175 -
accuracy: 0.9941 - val_loss: 0.0371 - val_accuracy: 0.9902
Epoch 30/400
937/937 [=====] - 6s 7ms/step - loss: 0.0187 -
accuracy: 0.9939 - val_loss: 0.0403 - val_accuracy: 0.9888
Epoch 31/400
937/937 [=====] - 6s 6ms/step - loss: 0.0188 -
accuracy: 0.9942 - val_loss: 0.0495 - val_accuracy: 0.9875
Epoch 32/400
937/937 [=====] - 6s 6ms/step - loss: 0.0165 -
accuracy: 0.9944 - val_loss: 0.0426 - val_accuracy: 0.9889
Epoch 33/400
937/937 [=====] - 6s 6ms/step - loss: 0.0160 -
accuracy: 0.9953 - val_loss: 0.0457 - val_accuracy: 0.9886
Restoring model weights from the end of the best epoch.
Epoch 00033: early stopping

```

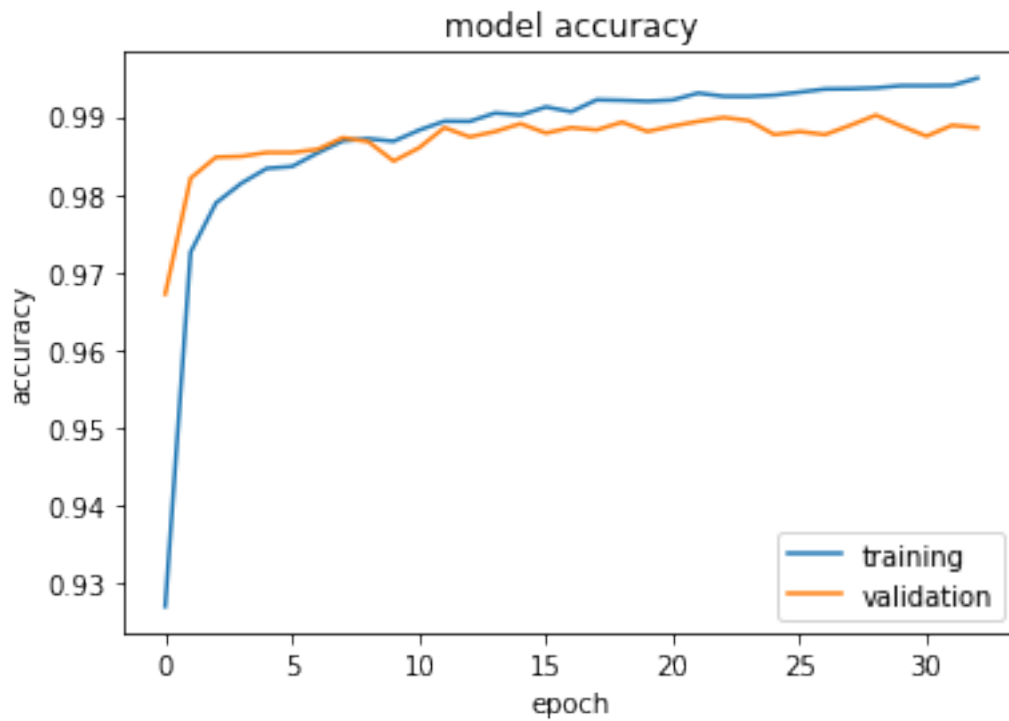
```

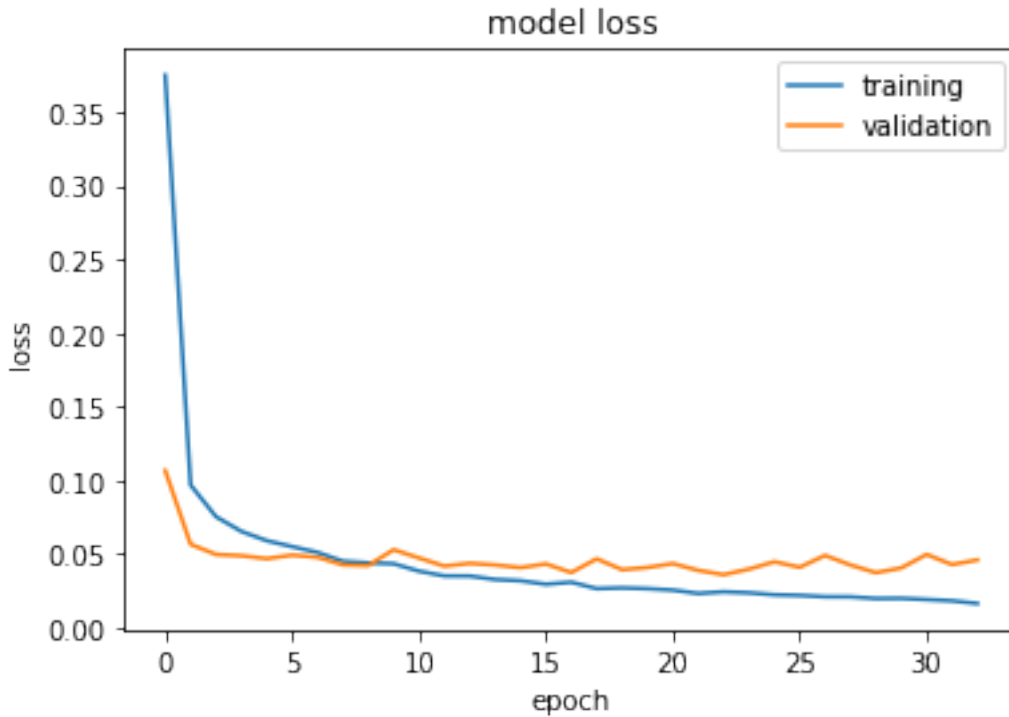
[ ]: plt.figure(1)
plt.plot(history1_un.history['accuracy'])
plt.plot(history1_un.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()

plt.figure(2)

```

```
plt.plot(history1_un.history['loss'])
plt.plot(history1_un.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```





This has similar performance to that of the pretrained model. Let's compare the plots more closely.

Observations and results

```
[ ]: plt.figure(1)
plt.plot(history1_after.history['accuracy'])
plt.plot(history1_un.history['accuracy'])
plt.title('training accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['pretrained_training', 'untrained_training'], loc='best')
plt.show()

plt.figure(2)
plt.plot(history1_after.history['val_accuracy'])
plt.plot(history1_un.history['val_accuracy'])
plt.title('validation accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['pretrained_validation', 'untrained_validation'], loc='best')
plt.show()

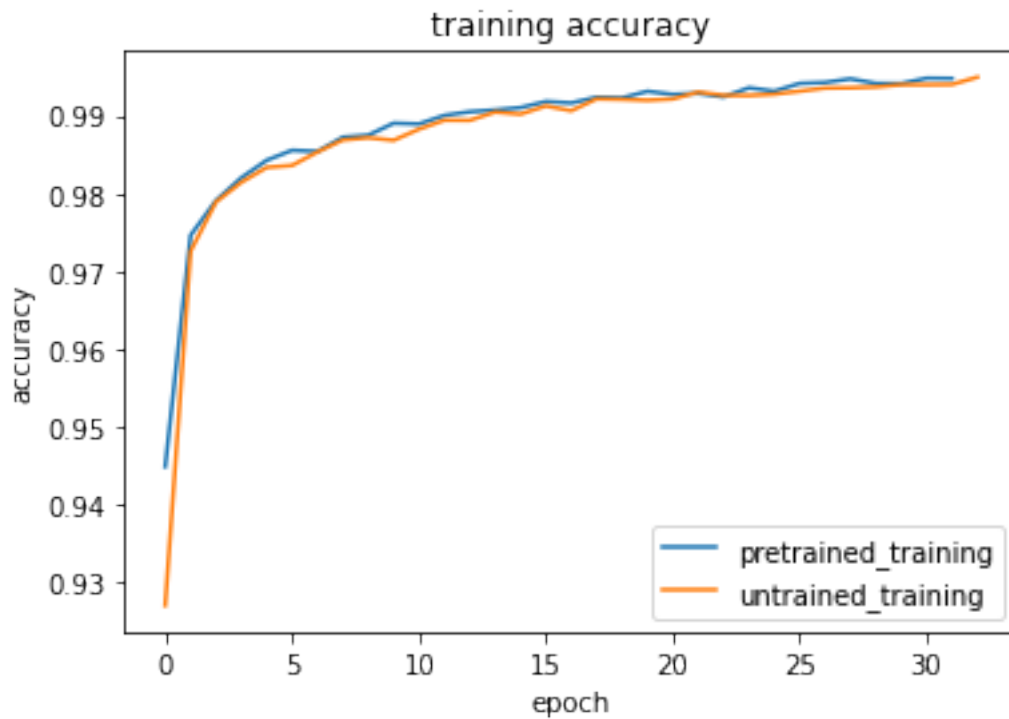
plt.figure(3)
plt.plot(history1_after.history['loss'])
```

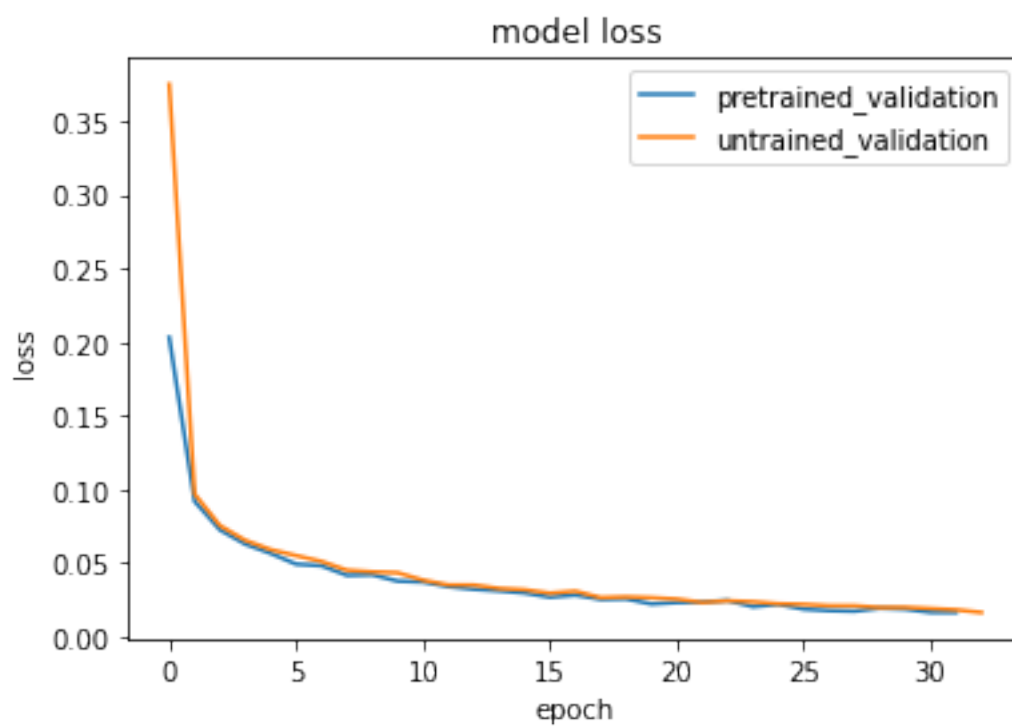
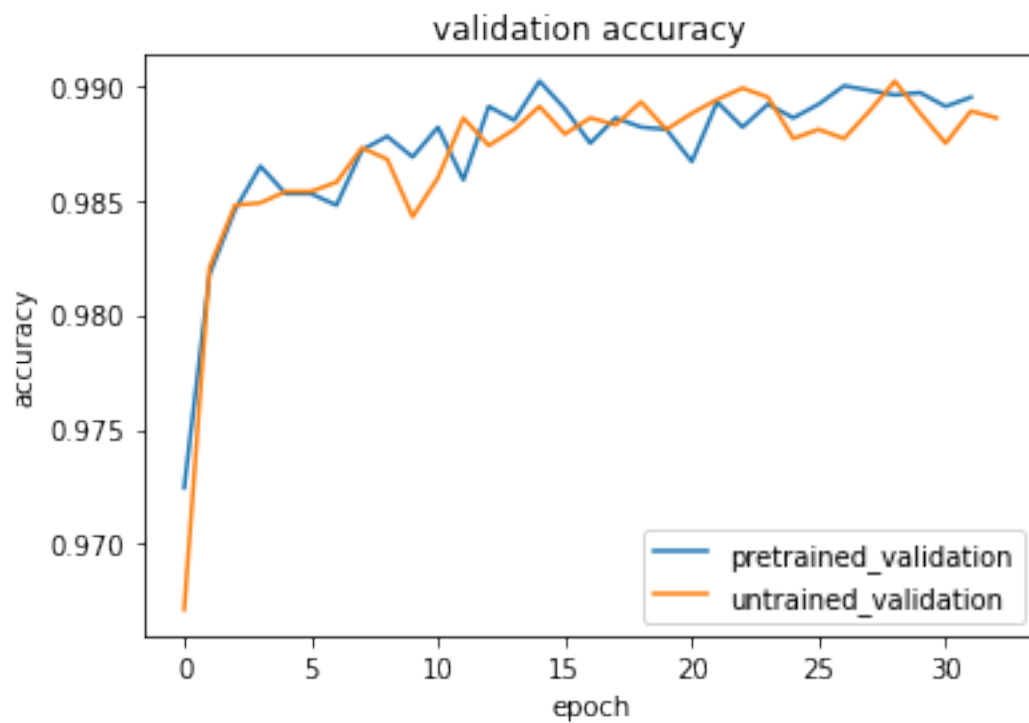
```

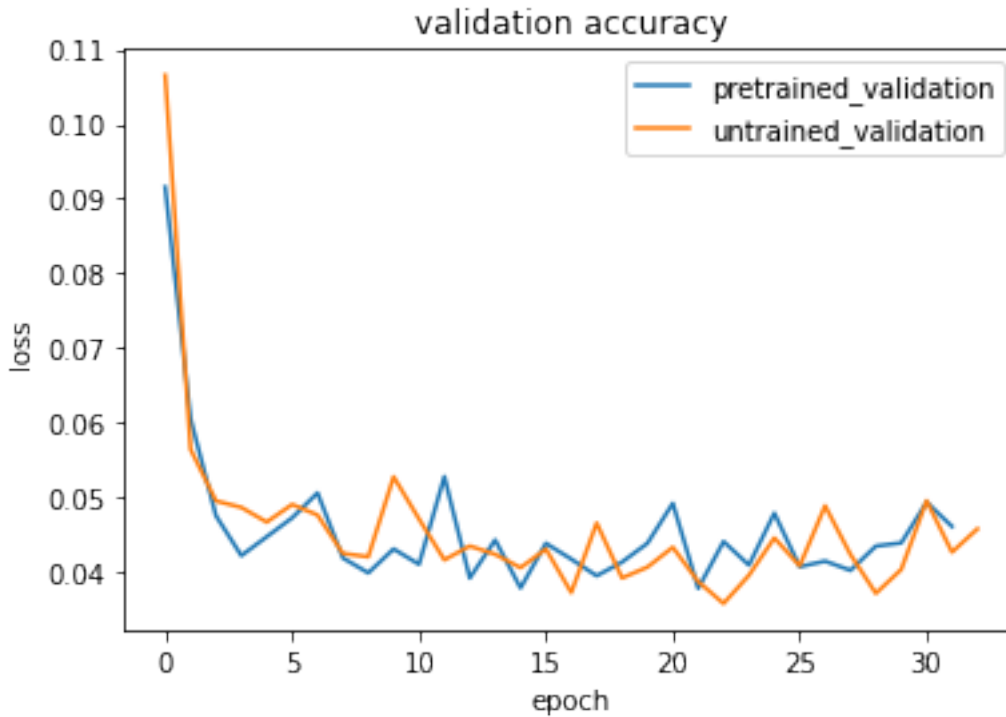
plt.plot(history1_un.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['pretrained_validation', 'untrained_validation'], loc='best')
plt.show()

plt.figure(4)
plt.plot(history1_after.history['val_loss'])
plt.plot(history1_un.history['val_loss'])
plt.title('validation accuracy')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['pretrained_validation', 'untrained_validation'], loc='best')
plt.show()

```







We can see that although both the models perform at-par with each other, the pretrained model performs ‘slightly’ better. The reasons are: - The pretrained accuracy for both the test and validation sets start higher than the randomly initialized models. - During the training phase, the pretrained overall performs better. - The same can be said for the loss, which starts lower for both the training and validation set for the pretrained model and for most part, stays lower. - **Overall, the pre-trained network has higher final accuracy and lower final loss and also has converged faster.**

1.4.2 2nd Architecture from Kaggle

Pre-training on our dataset

```
[ ]: model2 = Sequential()

# Lambda Layer for adding Padding
model2.add(Lambda(lambda image: tf.image.resize_with_crop_or_pad(
    image, 28, 28), input_shape=(*IMAGE_SIZE_BEFORE_PADDING, 1)))

# 1st Convolution Layer
model2.add(Conv2D(32, input_shape=(*IMAGE_SIZE, 1), kernel_size=3,
    ↪activation=mish))
model2.add(BatchNormalization())
model2.add(Conv2D(32, kernel_size=3, activation=mish))
model2.add(BatchNormalization())
```

```

model2.add(Conv2D(32, kernel_size=5, strides=2, padding='same',
    ↪activation=mish))
model2.add(BatchNormalization())
model2.add(Dropout(0.4))

# 2nd Convolution Layer
model2.add(Conv2D(64, kernel_size=3, activation=mish))
model2.add(BatchNormalization())
model2.add(Conv2D(64, kernel_size=3, activation=mish))
model2.add(BatchNormalization())
model2.add(Conv2D(64, kernel_size=5, strides=2, padding='same',
    ↪activation=mish))
model2.add(BatchNormalization())
model2.add(Dropout(0.4))

# 3rd Convolution Layer
model2.add(Conv2D(128, kernel_size = 4, activation=mish))
model2.add(BatchNormalization())

# Passing to a Fully Connected Layer
model2.add(Flatten())
model2.add(Dropout(0.4))

# Output Layer

# Increasing the softmax temperature
model2.add(Lambda(lambda x: x / temp))
model2.add(Dense(10, activation='softmax'))

model2.summary()

model2.compile(loss='categorical_crossentropy', optimizer=Adam(),
    ↪metrics=['accuracy'])

```

Model: "sequential_16"

Layer (type)	Output Shape	Param #
=====		
lambda_32 (Lambda)	(None, 28, 28, 1)	0

conv2d_67 (Conv2D)	(None, 26, 26, 32)	320

batch_normalization_85 (Batc	(None, 26, 26, 32)	128

conv2d_68 (Conv2D)	(None, 24, 24, 32)	9248

batch_normalization_86 (Batc	(None, 24, 24, 32)	128

conv2d_69 (Conv2D)	(None, 12, 12, 32)	25632
batch_normalization_87 (Batch Normalization)	(None, 12, 12, 32)	128
dropout_39 (Dropout)	(None, 12, 12, 32)	0
conv2d_70 (Conv2D)	(None, 10, 10, 64)	18496
batch_normalization_88 (Batch Normalization)	(None, 10, 10, 64)	256
conv2d_71 (Conv2D)	(None, 8, 8, 64)	36928
batch_normalization_89 (Batch Normalization)	(None, 8, 8, 64)	256
conv2d_72 (Conv2D)	(None, 4, 4, 64)	102464
batch_normalization_90 (Batch Normalization)	(None, 4, 4, 64)	256
dropout_40 (Dropout)	(None, 4, 4, 64)	0
conv2d_73 (Conv2D)	(None, 1, 1, 128)	131200
batch_normalization_91 (Batch Normalization)	(None, 1, 1, 128)	512
flatten_16 (Flatten)	(None, 128)	0
dropout_41 (Dropout)	(None, 128)	0
lambda_33 (Lambda)	(None, 128)	0
dense_34 (Dense)	(None, 10)	1290

=====
Total params: 327,242
Trainable params: 326,410
Non-trainable params: 832
=====

```
[ ]: checkpoint_filepath2 = 'part2_pretrained2/checkpoint'
model_checkpoint_callback2 = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath2,
    save_weights_only=True,
    monitor='val_loss',
    mode='min',
    save_best_only=True)

history2 = model2.fit(
```

```

train_generator,
epochs=EPOCHS,
validation_data=validation_generator,
steps_per_epoch = train_generator.samples // BATCH_SIZE,
validation_steps = validation_generator.samples // BATCH_SIZE,
callbacks=[model_checkpoint_callback2, early_stopping_callback2]
)

```

Epoch 1/400

5/5 [=====] - 7s 1s/step - loss: 2.3138 - accuracy: 0.1182 - val_loss: 2.3023 - val_accuracy: 0.1250

Epoch 2/400

5/5 [=====] - 5s 1s/step - loss: 2.1094 - accuracy: 0.2765 - val_loss: 2.3019 - val_accuracy: 0.1094

Epoch 3/400

5/5 [=====] - 5s 1s/step - loss: 1.8971 - accuracy: 0.4526 - val_loss: 2.3014 - val_accuracy: 0.0938

Epoch 4/400

5/5 [=====] - 5s 1s/step - loss: 1.7170 - accuracy: 0.5365 - val_loss: 2.2995 - val_accuracy: 0.1094

Epoch 5/400

5/5 [=====] - 5s 1s/step - loss: 1.5864 - accuracy: 0.6230 - val_loss: 2.2963 - val_accuracy: 0.1094

Epoch 6/400

5/5 [=====] - 5s 1s/step - loss: 1.4425 - accuracy: 0.6797 - val_loss: 2.3012 - val_accuracy: 0.1094

Epoch 7/400

5/5 [=====] - 5s 1s/step - loss: 1.3561 - accuracy: 0.6646 - val_loss: 2.2970 - val_accuracy: 0.1250

Epoch 8/400

5/5 [=====] - 5s 1s/step - loss: 1.2197 - accuracy: 0.7739 - val_loss: 2.3238 - val_accuracy: 0.0781

Epoch 9/400

5/5 [=====] - 5s 1s/step - loss: 1.0814 - accuracy: 0.7930 - val_loss: 2.3032 - val_accuracy: 0.1250

Epoch 10/400

5/5 [=====] - 5s 1s/step - loss: 1.0078 - accuracy: 0.8230 - val_loss: 2.3186 - val_accuracy: 0.0625

Epoch 11/400

5/5 [=====] - 5s 1s/step - loss: 0.9029 - accuracy: 0.8525 - val_loss: 2.3191 - val_accuracy: 0.1250

Epoch 12/400

5/5 [=====] - 5s 1s/step - loss: 0.8383 - accuracy: 0.8854 - val_loss: 2.3288 - val_accuracy: 0.1250

Epoch 13/400

5/5 [=====] - 5s 1s/step - loss: 0.7853 - accuracy: 0.8871 - val_loss: 2.3276 - val_accuracy: 0.0781

Epoch 14/400
5/5 [=====] - 5s 1s/step - loss: 0.6906 - accuracy: 0.9464 - val_loss: 2.3178 - val_accuracy: 0.1094

Epoch 15/400
5/5 [=====] - 5s 1s/step - loss: 0.6484 - accuracy: 0.9485 - val_loss: 2.3706 - val_accuracy: 0.1094

Epoch 16/400
5/5 [=====] - 5s 1s/step - loss: 0.5745 - accuracy: 0.9457 - val_loss: 2.4311 - val_accuracy: 0.0938

Epoch 17/400
5/5 [=====] - 5s 1s/step - loss: 0.5378 - accuracy: 0.9681 - val_loss: 2.4543 - val_accuracy: 0.1250

Epoch 18/400
5/5 [=====] - 5s 1s/step - loss: 0.4636 - accuracy: 0.9917 - val_loss: 2.4655 - val_accuracy: 0.1094

Epoch 19/400
5/5 [=====] - 5s 1s/step - loss: 0.4483 - accuracy: 0.9838 - val_loss: 2.6810 - val_accuracy: 0.0938

Epoch 20/400
5/5 [=====] - 5s 1s/step - loss: 0.4116 - accuracy: 0.9829 - val_loss: 2.6248 - val_accuracy: 0.1250

Epoch 21/400
5/5 [=====] - 5s 1s/step - loss: 0.3498 - accuracy: 1.0000 - val_loss: 2.6185 - val_accuracy: 0.1250

Epoch 22/400
5/5 [=====] - 5s 1s/step - loss: 0.3386 - accuracy: 0.9897 - val_loss: 2.7297 - val_accuracy: 0.1094

Epoch 23/400
5/5 [=====] - 5s 1s/step - loss: 0.2743 - accuracy: 0.9990 - val_loss: 2.7349 - val_accuracy: 0.1094

Epoch 24/400
5/5 [=====] - 5s 1s/step - loss: 0.2714 - accuracy: 0.9897 - val_loss: 2.7203 - val_accuracy: 0.1094

Epoch 25/400
5/5 [=====] - 5s 1s/step - loss: 0.2513 - accuracy: 1.0000 - val_loss: 2.8316 - val_accuracy: 0.1094

Epoch 26/400
5/5 [=====] - 5s 1s/step - loss: 0.2315 - accuracy: 1.0000 - val_loss: 2.7348 - val_accuracy: 0.1250

Epoch 27/400
5/5 [=====] - 5s 1s/step - loss: 0.2211 - accuracy: 0.9961 - val_loss: 2.8925 - val_accuracy: 0.1094

Epoch 28/400
5/5 [=====] - 5s 1s/step - loss: 0.2223 - accuracy: 1.0000 - val_loss: 2.8186 - val_accuracy: 0.1094

Epoch 29/400
5/5 [=====] - 5s 1s/step - loss: 0.1764 - accuracy: 1.0000 - val_loss: 2.9625 - val_accuracy: 0.0938

Epoch 30/400
5/5 [=====] - 5s 1s/step - loss: 0.1890 - accuracy: 1.0000 - val_loss: 2.9418 - val_accuracy: 0.1094

Epoch 31/400
5/5 [=====] - 5s 1s/step - loss: 0.1569 - accuracy: 1.0000 - val_loss: 3.0047 - val_accuracy: 0.1094

Epoch 32/400
5/5 [=====] - 5s 1s/step - loss: 0.1630 - accuracy: 1.0000 - val_loss: 2.9697 - val_accuracy: 0.0938

Epoch 33/400
5/5 [=====] - 5s 1s/step - loss: 0.1461 - accuracy: 0.9990 - val_loss: 2.9317 - val_accuracy: 0.1094

Epoch 34/400
5/5 [=====] - 5s 1s/step - loss: 0.1385 - accuracy: 1.0000 - val_loss: 3.0667 - val_accuracy: 0.1094

Epoch 35/400
5/5 [=====] - 5s 1s/step - loss: 0.1221 - accuracy: 1.0000 - val_loss: 2.9470 - val_accuracy: 0.0938

Epoch 36/400
5/5 [=====] - 5s 1s/step - loss: 0.1207 - accuracy: 1.0000 - val_loss: 2.9738 - val_accuracy: 0.1250

Epoch 37/400
5/5 [=====] - 5s 1s/step - loss: 0.1099 - accuracy: 1.0000 - val_loss: 3.0589 - val_accuracy: 0.1406

Epoch 38/400
5/5 [=====] - 5s 1s/step - loss: 0.0955 - accuracy: 1.0000 - val_loss: 3.0744 - val_accuracy: 0.1250

Epoch 39/400
5/5 [=====] - 5s 1s/step - loss: 0.1074 - accuracy: 1.0000 - val_loss: 3.1594 - val_accuracy: 0.1250

Epoch 40/400
5/5 [=====] - 5s 1s/step - loss: 0.0911 - accuracy: 1.0000 - val_loss: 3.1274 - val_accuracy: 0.1250

Epoch 41/400
5/5 [=====] - 5s 1s/step - loss: 0.0868 - accuracy: 1.0000 - val_loss: 3.0317 - val_accuracy: 0.1094

Epoch 42/400
5/5 [=====] - 5s 1s/step - loss: 0.0772 - accuracy: 1.0000 - val_loss: 3.1888 - val_accuracy: 0.0938

Epoch 43/400
5/5 [=====] - 5s 1s/step - loss: 0.0826 - accuracy: 1.0000 - val_loss: 3.1052 - val_accuracy: 0.1094

Epoch 44/400
5/5 [=====] - 5s 1s/step - loss: 0.0763 - accuracy: 1.0000 - val_loss: 3.1776 - val_accuracy: 0.0938

Epoch 45/400
5/5 [=====] - 5s 1s/step - loss: 0.0668 - accuracy: 1.0000 - val_loss: 3.3263 - val_accuracy: 0.0938

```

Epoch 46/400
5/5 [=====] - 5s 1s/step - loss: 0.0709 - accuracy:
1.0000 - val_loss: 3.2739 - val_accuracy: 0.1094
Epoch 47/400
5/5 [=====] - 5s 1s/step - loss: 0.0714 - accuracy:
1.0000 - val_loss: 3.0664 - val_accuracy: 0.1250
Epoch 48/400
5/5 [=====] - 5s 1s/step - loss: 0.0720 - accuracy:
0.9961 - val_loss: 3.0611 - val_accuracy: 0.1094
Epoch 49/400
5/5 [=====] - 5s 1s/step - loss: 0.0548 - accuracy:
1.0000 - val_loss: 3.0912 - val_accuracy: 0.1250
Epoch 50/400
5/5 [=====] - 5s 1s/step - loss: 0.0572 - accuracy:
1.0000 - val_loss: 3.0163 - val_accuracy: 0.1406
Epoch 51/400
5/5 [=====] - 5s 1s/step - loss: 0.0587 - accuracy:
1.0000 - val_loss: 3.1670 - val_accuracy: 0.1406
Epoch 52/400
5/5 [=====] - 5s 1s/step - loss: 0.0546 - accuracy:
1.0000 - val_loss: 3.1523 - val_accuracy: 0.1250
Epoch 53/400
5/5 [=====] - 5s 1s/step - loss: 0.0555 - accuracy:
1.0000 - val_loss: 3.2594 - val_accuracy: 0.0938
Epoch 54/400
5/5 [=====] - 5s 1s/step - loss: 0.0506 - accuracy:
1.0000 - val_loss: 3.2338 - val_accuracy: 0.0938
Epoch 55/400
5/5 [=====] - 5s 1s/step - loss: 0.0495 - accuracy:
1.0000 - val_loss: 3.2775 - val_accuracy: 0.0938
Restoring model weights from the end of the best epoch.
Epoch 00055: early stopping

```

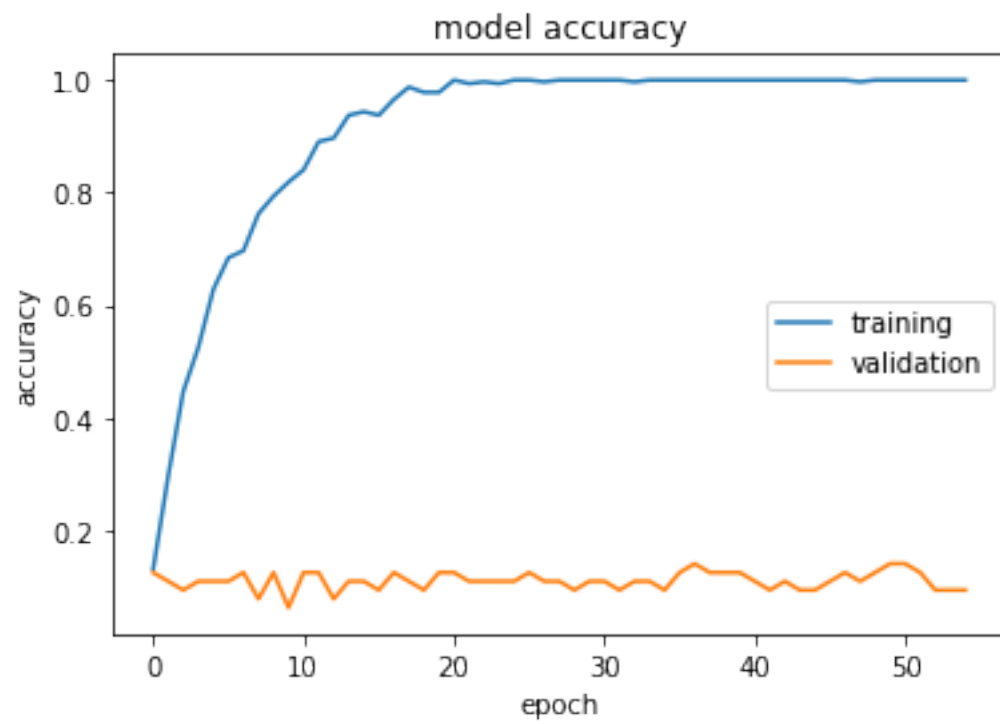
```

[ ]: plt.figure(1)
plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()

plt.figure(2)
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')

```

```
plt.xlabel('epoch')  
plt.legend(['training', 'validation'], loc='best')  
plt.show()
```





This actually performs worse in pre-training, than the modified LeNet used before.

I think the reason for this might be that this model has more parameters and is more complex so, it does not do well with these small training set.

Let's see how it performs on the MNIST dataset.

Training on MNIST with pre-trained weights

```
[ ]: model2 = Sequential()

# Lambda Layer for adding Padding
model2.add(Lambda(lambda image: tf.image.resize_with_crop_or_pad(
    image, 28, 28), input_shape=(*IMAGE_SIZE_BEFORE_PADDING, 1)))

# 1st Convolution Layer
model2.add(Conv2D(32, input_shape=(*IMAGE_SIZE, 1), kernel_size=3,
    ↪activation=mish))
model2.add(BatchNormalization())
model2.add(Conv2D(32, kernel_size=3, activation=mish))
model2.add(BatchNormalization())
model2.add(Conv2D(32, kernel_size=5, strides=2, padding='same',
    ↪activation=mish))
model2.add(BatchNormalization())
model2.add(Dropout(0.4))
```

```

# 2nd Convolution Layer
model2.add(Conv2D(64, kernel_size=3, activation=mish))
model2.add(BatchNormalization())
model2.add(Conv2D(64, kernel_size=3, activation=mish))
model2.add(BatchNormalization())
model2.add(Conv2D(64, kernel_size=5, strides=2, padding='same',
    ↪activation=mish))
model2.add(BatchNormalization())
model2.add(Dropout(0.4))

# 3rd Convolution Layer
model2.add(Conv2D(128, kernel_size = 4, activation=mish))
model2.add(BatchNormalization())

# Passing to a Fully Connected Layer
model2.add(Flatten())
model2.add(Dropout(0.4))

# Output Layer

# Increasing the softmax temperature
model2.add(Lambda(lambda x: x / temp))
model2.add(Dense(10, activation='softmax'))

model2.summary()

model2.load_weights('part2_pretrained2/checkpoint')

model2.compile(loss='categorical_crossentropy', optimizer=Adam(),
    ↪metrics=['accuracy'])

```

Model: "sequential_18"

Layer (type)	Output Shape	Param #
=====		
lambda_36 (Lambda)	(None, 28, 28, 1)	0

conv2d_81 (Conv2D)	(None, 26, 26, 32)	320

batch_normalization_99 (Batch Normalization)	(None, 26, 26, 32)	128

conv2d_82 (Conv2D)	(None, 24, 24, 32)	9248

batch_normalization_100 (Batch Normalization)	(None, 24, 24, 32)	128

conv2d_83 (Conv2D)	(None, 12, 12, 32)	25632


```

-----
batch_normalization_101 (Bat (None, 12, 12, 32)      128
-----
dropout_45 (Dropout)      (None, 12, 12, 32)      0
-----
conv2d_84 (Conv2D)      (None, 10, 10, 64)      18496
-----
batch_normalization_102 (Bat (None, 10, 10, 64)      256
-----
conv2d_85 (Conv2D)      (None, 8, 8, 64)      36928
-----
batch_normalization_103 (Bat (None, 8, 8, 64)      256
-----
conv2d_86 (Conv2D)      (None, 4, 4, 64)      102464
-----
batch_normalization_104 (Bat (None, 4, 4, 64)      256
-----
dropout_46 (Dropout)      (None, 4, 4, 64)      0
-----
conv2d_87 (Conv2D)      (None, 1, 1, 128)      131200
-----
batch_normalization_105 (Bat (None, 1, 1, 128)      512
-----
flatten_18 (Flatten)      (None, 128)      0
-----
dropout_47 (Dropout)      (None, 128)      0
-----
lambda_37 (Lambda)      (None, 128)      0
-----
dense_36 (Dense)      (None, 10)      1290
=====
Total params: 327,242
Trainable params: 326,410
Non-trainable params: 832
-----

```

```

[ ]: checkpoint_filepath2_after = 'part2_after_training2/checkpoint'
model_checkpoint_callback2_after = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath2_after,
    save_weights_only=True,
    monitor='val_loss',
    mode='min',
    save_best_only=True)

history2_after = model2.fit(
    mnist_train_gen,
    epochs=EPOCHS,

```

```

validation_data=mnist_test_gen,
steps_per_epoch = len(x_train) // BATCH_SIZE,
validation_steps = len(x_test) // BATCH_SIZE,
callbacks=[model_checkpoint_callback2_after, early_stopping_callback]
)

```

Epoch 1/400

937/937 [=====] - 12s 12ms/step - loss: 0.4861 - accuracy: 0.8868 - val_loss: 0.0475 - val_accuracy: 0.9872

Epoch 2/400

937/937 [=====] - 11s 11ms/step - loss: 0.0653 - accuracy: 0.9817 - val_loss: 0.0325 - val_accuracy: 0.9902

Epoch 3/400

937/937 [=====] - 11s 11ms/step - loss: 0.0515 - accuracy: 0.9841 - val_loss: 0.0401 - val_accuracy: 0.9872

Epoch 4/400

937/937 [=====] - 11s 12ms/step - loss: 0.0401 - accuracy: 0.9874 - val_loss: 0.0331 - val_accuracy: 0.9895

Epoch 5/400

937/937 [=====] - 11s 11ms/step - loss: 0.0385 - accuracy: 0.9884 - val_loss: 0.0253 - val_accuracy: 0.9927

Epoch 6/400

937/937 [=====] - 11s 12ms/step - loss: 0.0311 - accuracy: 0.9906 - val_loss: 0.0341 - val_accuracy: 0.9894

Epoch 7/400

937/937 [=====] - 11s 12ms/step - loss: 0.0308 - accuracy: 0.9904 - val_loss: 0.0228 - val_accuracy: 0.9931

Epoch 8/400

937/937 [=====] - 11s 12ms/step - loss: 0.0268 - accuracy: 0.9917 - val_loss: 0.0269 - val_accuracy: 0.9923

Epoch 9/400

937/937 [=====] - 11s 12ms/step - loss: 0.0247 - accuracy: 0.9921 - val_loss: 0.0303 - val_accuracy: 0.9904

Epoch 10/400

937/937 [=====] - 11s 11ms/step - loss: 0.0201 - accuracy: 0.9939 - val_loss: 0.0268 - val_accuracy: 0.9926

Epoch 11/400

937/937 [=====] - 11s 12ms/step - loss: 0.0205 - accuracy: 0.9939 - val_loss: 0.0261 - val_accuracy: 0.9918

Epoch 12/400

937/937 [=====] - 11s 12ms/step - loss: 0.0215 - accuracy: 0.9933 - val_loss: 0.0250 - val_accuracy: 0.9930

Epoch 13/400

937/937 [=====] - 11s 11ms/step - loss: 0.0203 - accuracy: 0.9939 - val_loss: 0.0249 - val_accuracy: 0.9928

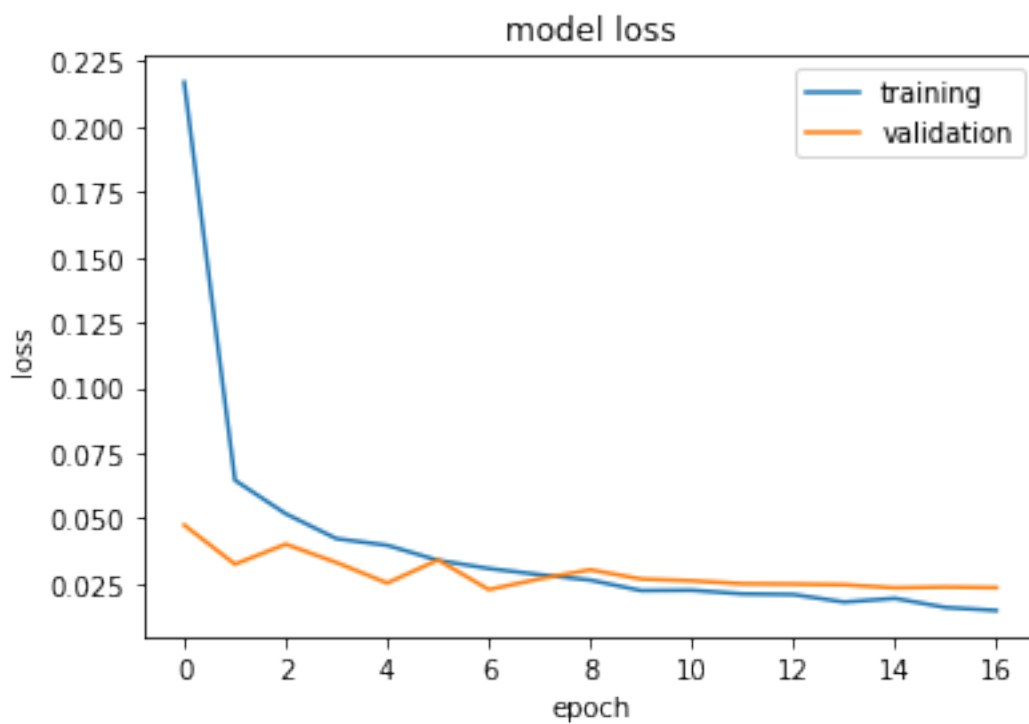
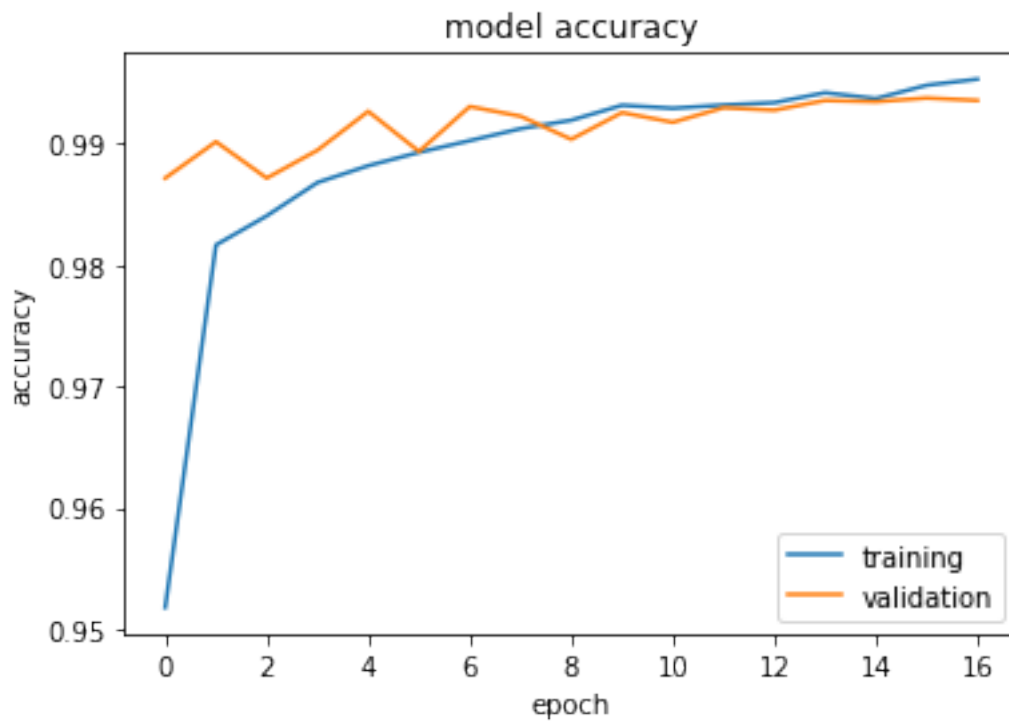
Epoch 14/400

937/937 [=====] - 11s 11ms/step - loss: 0.0192 -

```
accuracy: 0.9939 - val_loss: 0.0246 - val_accuracy: 0.9936
Epoch 15/400
937/937 [=====] - 11s 12ms/step - loss: 0.0170 -
accuracy: 0.9946 - val_loss: 0.0234 - val_accuracy: 0.9935
Epoch 16/400
937/937 [=====] - 11s 11ms/step - loss: 0.0146 -
accuracy: 0.9951 - val_loss: 0.0237 - val_accuracy: 0.9938
Epoch 17/400
937/937 [=====] - 11s 12ms/step - loss: 0.0147 -
accuracy: 0.9951 - val_loss: 0.0235 - val_accuracy: 0.9936
Restoring model weights from the end of the best epoch.
Epoch 00017: early stopping
```

```
[ ]: plt.figure(1)
plt.plot(history2_after.history['accuracy'])
plt.plot(history2_after.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()

plt.figure(2)
plt.plot(history2_after.history['loss'])
plt.plot(history2_after.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```



This is pretty good! It achieves higher accuracy than the LeNet, and in almost half the time.

Untrained network with random initializations

```
[ ]: model2 = Sequential()

# Lambda Layer for adding Padding
model2.add(Lambda(lambda image: tf.image.resize_with_crop_or_pad(
    image, 28, 28), input_shape=(*IMAGE_SIZE_BEFORE_PADDING, 1)))

# 1st Convolution Layer
model2.add(Conv2D(32, input_shape=(*IMAGE_SIZE, 1), kernel_size=3,
    ↪activation=mish))
model2.add(BatchNormalization())
model2.add(Conv2D(32, kernel_size=3, activation=mish))
model2.add(BatchNormalization())
model2.add(Conv2D(32, kernel_size=5, strides=2, padding='same',
    ↪activation=mish))
model2.add(BatchNormalization())
model2.add(Dropout(0.4))

# 2nd Convolution Layer
model2.add(Conv2D(64, kernel_size=3, activation=mish))
model2.add(BatchNormalization())
model2.add(Conv2D(64, kernel_size=3, activation=mish))
model2.add(BatchNormalization())
model2.add(Conv2D(64, kernel_size=5, strides=2, padding='same',
    ↪activation=mish))
model2.add(BatchNormalization())
model2.add(Dropout(0.4))

# 3rd Convolution Layer
model2.add(Conv2D(128, kernel_size = 4, activation=mish))
model2.add(BatchNormalization())

# Passing to a Fully Connected Layer
model2.add(Flatten())
model2.add(Dropout(0.4))

# Output Layer

# Increasing the softmax temperature
model2.add(Lambda(lambda x: x / temp))
model2.add(Dense(10, activation='softmax'))

model2.summary()
```

```
model2.compile(loss='categorical_crossentropy', optimizer=Adam(),  
↳metrics=['accuracy'])
```

Model: "sequential_20"

Layer (type)	Output Shape	Param #
lambda_40 (Lambda)	(None, 28, 28, 1)	0
conv2d_95 (Conv2D)	(None, 26, 26, 32)	320
batch_normalization_113 (Batch Normalization)	(None, 26, 26, 32)	128
conv2d_96 (Conv2D)	(None, 24, 24, 32)	9248
batch_normalization_114 (Batch Normalization)	(None, 24, 24, 32)	128
conv2d_97 (Conv2D)	(None, 12, 12, 32)	25632
batch_normalization_115 (Batch Normalization)	(None, 12, 12, 32)	128
dropout_51 (Dropout)	(None, 12, 12, 32)	0
conv2d_98 (Conv2D)	(None, 10, 10, 64)	18496
batch_normalization_116 (Batch Normalization)	(None, 10, 10, 64)	256
conv2d_99 (Conv2D)	(None, 8, 8, 64)	36928
batch_normalization_117 (Batch Normalization)	(None, 8, 8, 64)	256
conv2d_100 (Conv2D)	(None, 4, 4, 64)	102464
batch_normalization_118 (Batch Normalization)	(None, 4, 4, 64)	256
dropout_52 (Dropout)	(None, 4, 4, 64)	0
conv2d_101 (Conv2D)	(None, 1, 1, 128)	131200
batch_normalization_119 (Batch Normalization)	(None, 1, 1, 128)	512
flatten_20 (Flatten)	(None, 128)	0
dropout_53 (Dropout)	(None, 128)	0
lambda_41 (Lambda)	(None, 128)	0

```
dense_38 (Dense)                (None, 10)                1290
=====
Total params: 327,242
Trainable params: 326,410
Non-trainable params: 832
-----
```

```
[ ]: checkpoint_filepath2_un = 'part2_untrained2/checkpoint'
model_checkpoint_callback2_un = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath2_un,
    save_weights_only=True,
    monitor='val_loss',
    mode='min',
    save_best_only=True)

history2_un = model2.fit(
    train_generator3,
    epochs=17,
    validation_data=mnist_test_gen,
    steps_per_epoch = len(x_train) // BATCH_SIZE,
    validation_steps = len(x_test) // BATCH_SIZE,
    callbacks=[model_checkpoint_callback2_un, early_stopping_callback]
)
```

```
Epoch 1/17
937/937 [=====] - 12s 12ms/step - loss: 0.4512 -
accuracy: 0.9017 - val_loss: 0.0512 - val_accuracy: 0.9836
Epoch 2/17
937/937 [=====] - 11s 11ms/step - loss: 0.0633 -
accuracy: 0.9817 - val_loss: 0.0342 - val_accuracy: 0.9887
Epoch 3/17
937/937 [=====] - 11s 11ms/step - loss: 0.0458 -
accuracy: 0.9865 - val_loss: 0.0382 - val_accuracy: 0.9879
Epoch 4/17
937/937 [=====] - 11s 11ms/step - loss: 0.0421 -
accuracy: 0.9878 - val_loss: 0.0307 - val_accuracy: 0.9914
Epoch 5/17
937/937 [=====] - 11s 12ms/step - loss: 0.0341 -
accuracy: 0.9899 - val_loss: 0.0321 - val_accuracy: 0.9900
Epoch 6/17
937/937 [=====] - 11s 11ms/step - loss: 0.0332 -
accuracy: 0.9898 - val_loss: 0.0259 - val_accuracy: 0.9917
Epoch 7/17
937/937 [=====] - 11s 11ms/step - loss: 0.0294 -
accuracy: 0.9908 - val_loss: 0.0253 - val_accuracy: 0.9918
Epoch 8/17
937/937 [=====] - 11s 12ms/step - loss: 0.0294 -
accuracy: 0.9913 - val_loss: 0.0243 - val_accuracy: 0.9932
```

```

Epoch 9/17
937/937 [=====] - 11s 12ms/step - loss: 0.0257 -
accuracy: 0.9924 - val_loss: 0.0250 - val_accuracy: 0.9926
Epoch 10/17
937/937 [=====] - 11s 11ms/step - loss: 0.0210 -
accuracy: 0.9935 - val_loss: 0.0211 - val_accuracy: 0.9934
Epoch 11/17
937/937 [=====] - 11s 11ms/step - loss: 0.0212 -
accuracy: 0.9934 - val_loss: 0.0246 - val_accuracy: 0.9931
Epoch 12/17
937/937 [=====] - 11s 11ms/step - loss: 0.0193 -
accuracy: 0.9943 - val_loss: 0.0225 - val_accuracy: 0.9938
Epoch 13/17
937/937 [=====] - 11s 11ms/step - loss: 0.0190 -
accuracy: 0.9940 - val_loss: 0.0281 - val_accuracy: 0.9919
Epoch 14/17
937/937 [=====] - 11s 12ms/step - loss: 0.0200 -
accuracy: 0.9937 - val_loss: 0.0249 - val_accuracy: 0.9940
Epoch 15/17
937/937 [=====] - 11s 12ms/step - loss: 0.0134 -
accuracy: 0.9959 - val_loss: 0.0287 - val_accuracy: 0.9936
Epoch 16/17
937/937 [=====] - 11s 11ms/step - loss: 0.0169 -
accuracy: 0.9948 - val_loss: 0.0254 - val_accuracy: 0.9938
Epoch 17/17
937/937 [=====] - 11s 11ms/step - loss: 0.0148 -
accuracy: 0.9950 - val_loss: 0.0287 - val_accuracy: 0.9932

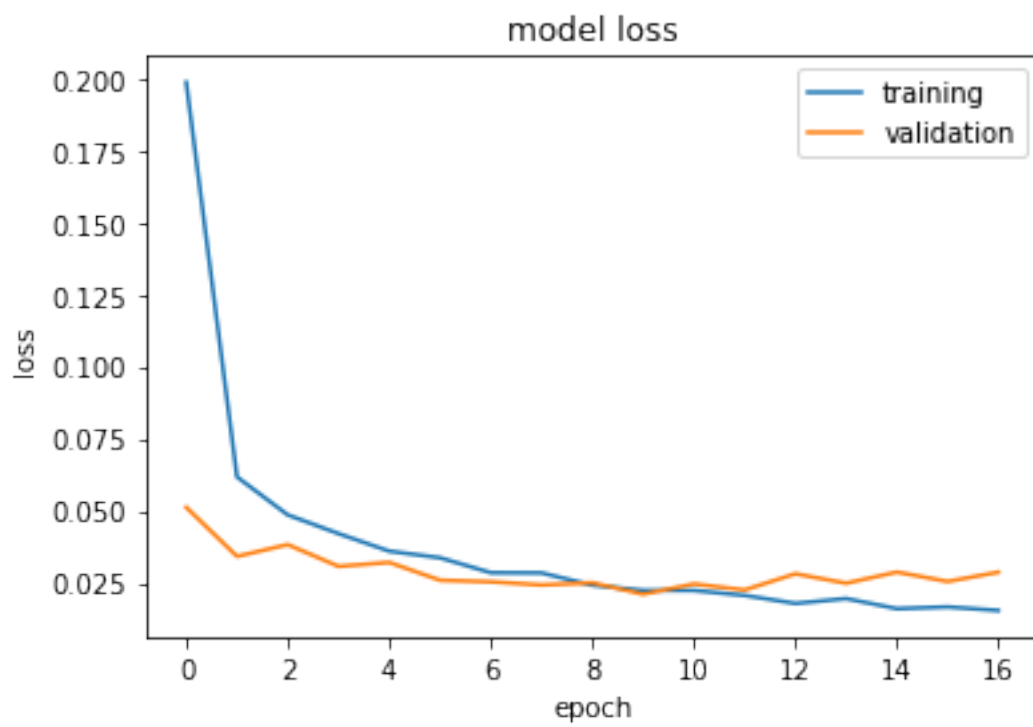
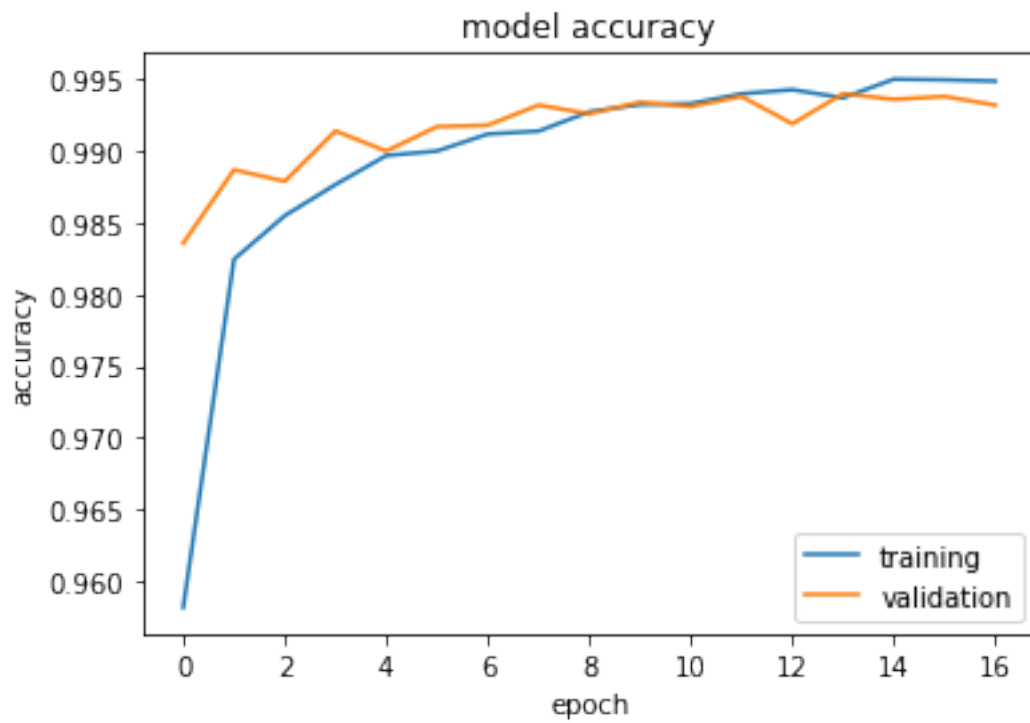
```

```

[ ]: plt.figure(1)
plt.plot(history2_un.history['accuracy'])
plt.plot(history2_un.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()

plt.figure(2)
plt.plot(history2_un.history['loss'])
plt.plot(history2_un.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()

```

This performs the same as before. Let's see them together to notice the difference.

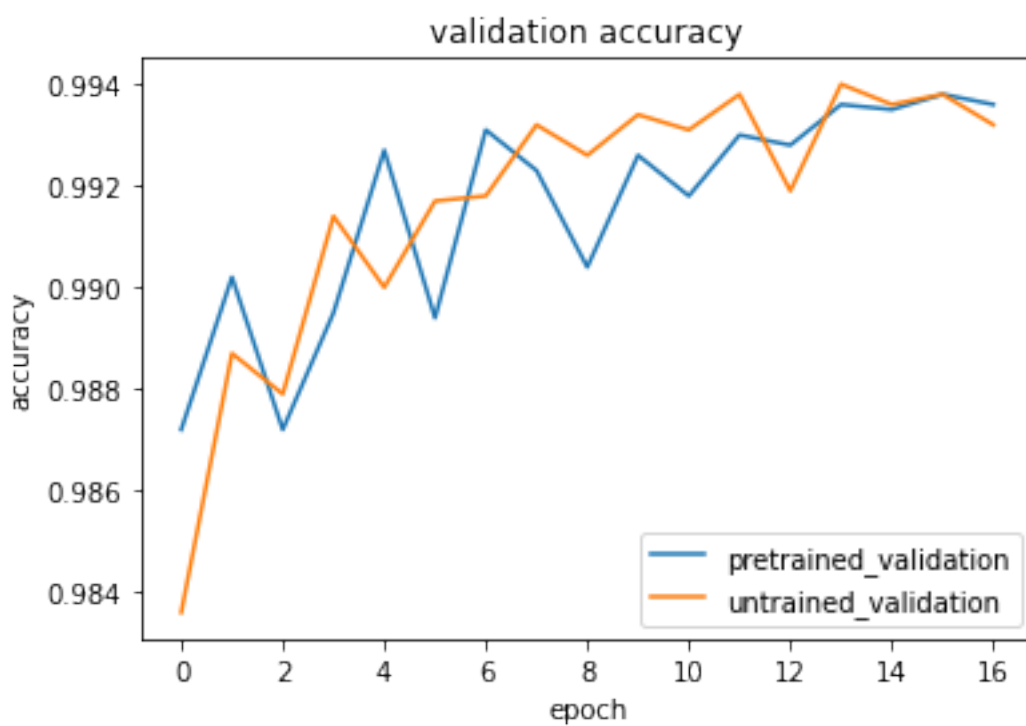
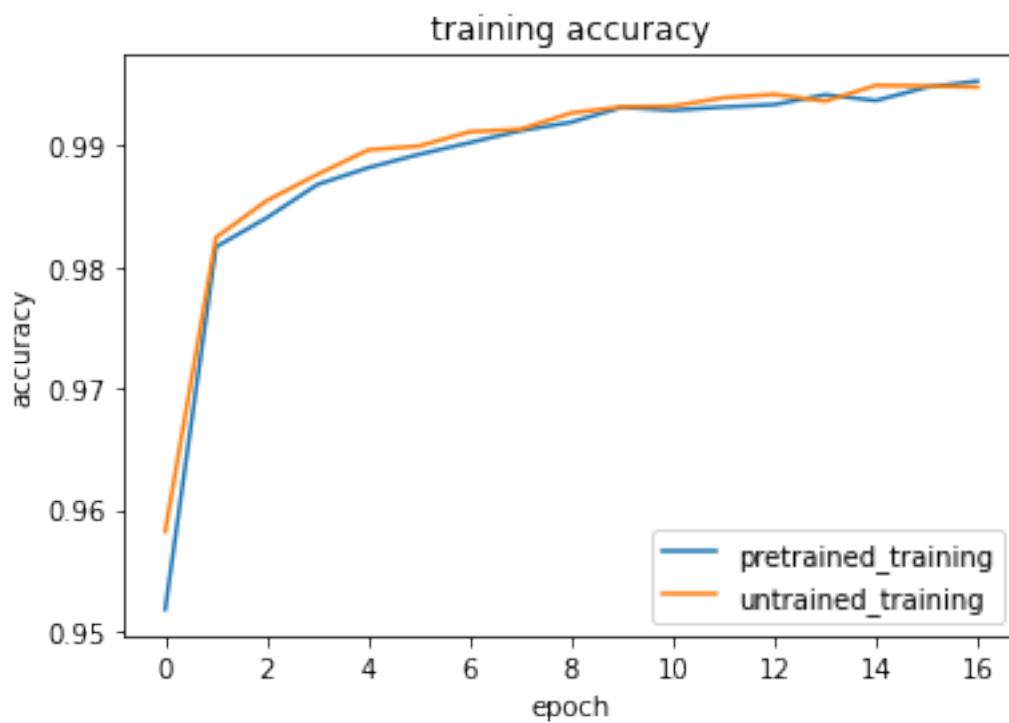
Observations and results

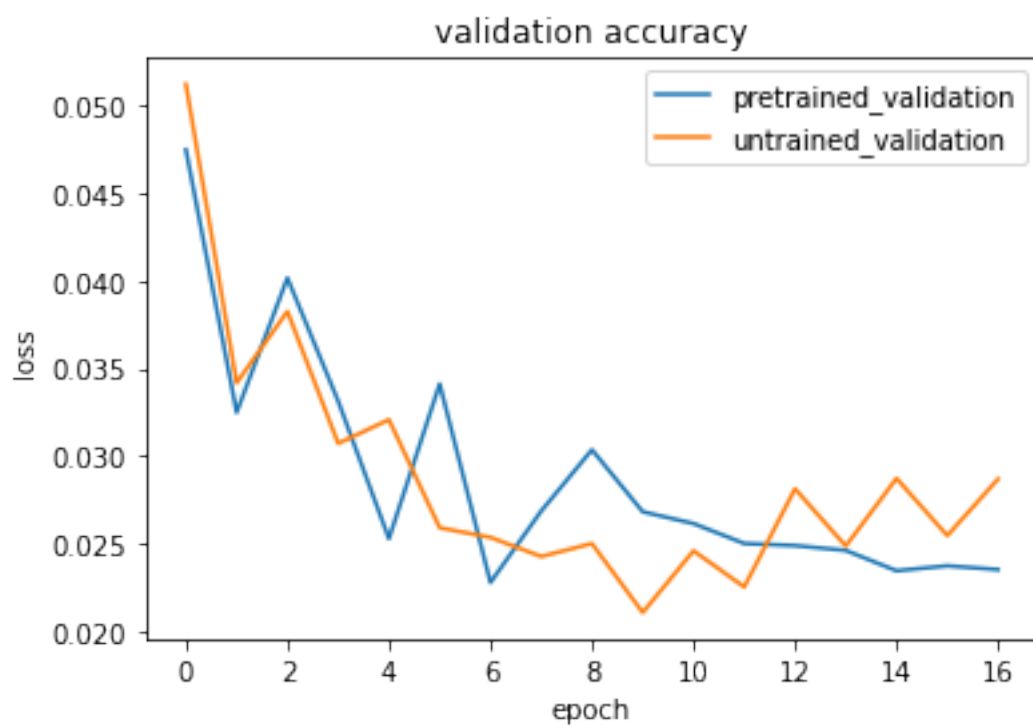
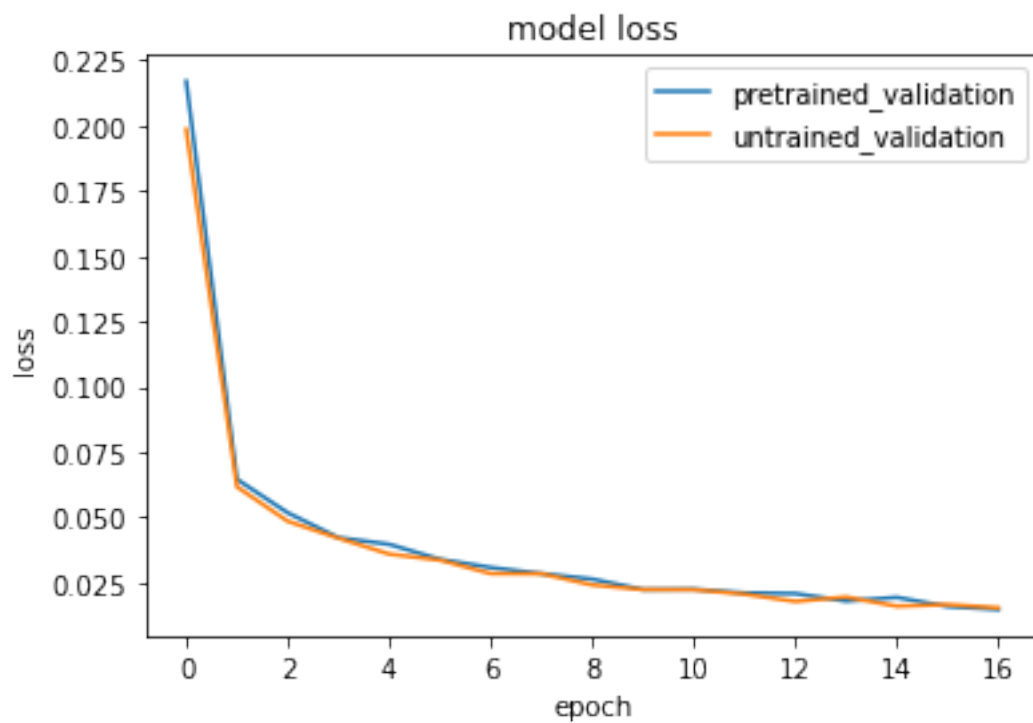
```
[ ]: plt.figure(1)
plt.plot(history2_after.history['accuracy'])
plt.plot(history2_un.history['accuracy'])
plt.title('training accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['pretrained_training', 'untrained_training'], loc='best')
plt.show()

plt.figure(2)
plt.plot(history2_after.history['val_accuracy'])
plt.plot(history2_un.history['val_accuracy'])
plt.title('validation accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['pretrained_validation', 'untrained_validation'], loc='best')
plt.show()

plt.figure(3)
plt.plot(history2_after.history['loss'])
plt.plot(history2_un.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['pretrained_validation', 'untrained_validation'], loc='best')
plt.show()

plt.figure(4)
plt.plot(history2_after.history['val_loss'])
plt.plot(history2_un.history['val_loss'])
plt.title('validation accuracy')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['pretrained_validation', 'untrained_validation'], loc='best')
plt.show()
```





This is interesting. For this case, the untrained model performs ‘slightly’ better than the pretrained one during the initial epochs but the pretrained model does better in the end.

As seen before, it did not perform well during pre-training. This may be due to the more complex nature of the architecture and less availability of the pretraining data.

It is to be noted that this architecture achieved very good accuracy in almost half the epochs of the original LeNet.

Some observations: - The untrained accuracy for both the test and validation sets start higher than the pretrained models. - During the training phase, the pretrained overall performs better in the end. - The same can be said for the loss, which starts lower for both the training and validation set for the pretrained model and for most part, stays lower. - **Overall, the pre-trained network has higher final accuracy and lower final loss.**

2 Task 2 Part 3

2.1 Extracting the dataset

```
[8]: shutil.unpack_archive('mnistTask3.zip', 'input/part3')
```

```
[9]: train_datagen3 = ImageDataGenerator(rescale=1./255)
```

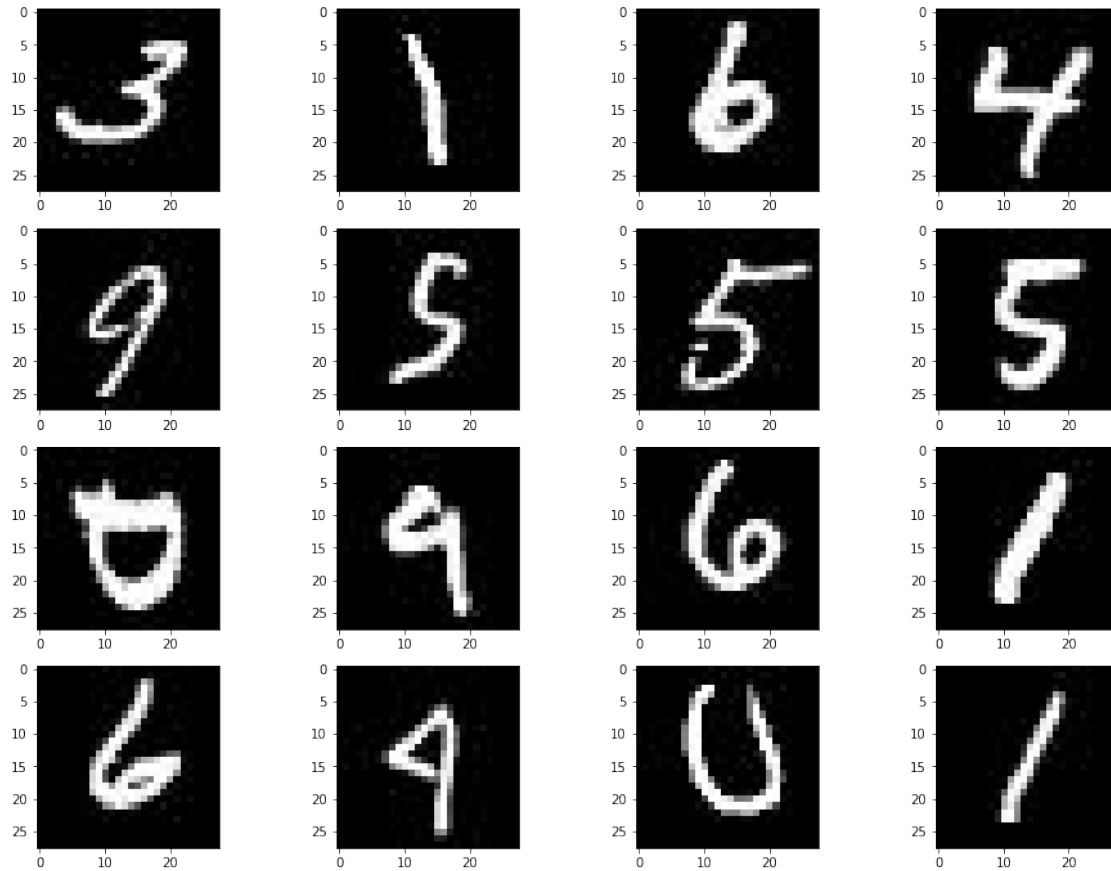
```
[10]: train_generator3 = train_datagen3.flow_from_directory(
        'input/part3/mnistTask',
        target_size=IMAGE_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        color_mode='grayscale',
        subset='training',
        seed=42,
        shuffle=True)

X_train_batch3, y_train_batch3 = train_generator3.next()
print(X_train_batch3.shape, y_train_batch3.shape)
print(y_train_batch3[0])
plt.figure(figsize=(16,12))
for i in range(1, 17):
    plt.subplot(4,4,i)
    imshow(tf.squeeze(X_train_batch3[i]), cmap='gray')
plt.show()
```

Found 60000 images belonging to 10 classes.

(64, 28, 28, 1) (64, 10)

[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]



2.2 Building the Model

Using Scratch Random Initialization I've used the 2nd Architecture for faster convergence.

```
[25]: model3 = Sequential()

# 1st Convolution Layer
model3.add(Conv2D(32, input_shape=(*IMAGE_SIZE, 1), kernel_size=3,
    ↪activation=mish))
model3.add(BatchNormalization())
model3.add(Conv2D(32, kernel_size=3, activation=mish))
model3.add(BatchNormalization())
model3.add(Conv2D(32, kernel_size=5, strides=2, padding='same',
    ↪activation=mish))
model3.add(BatchNormalization())
model3.add(Dropout(0.4))

# 2nd Convolution Layer
model3.add(Conv2D(64, kernel_size=3, activation=mish))
model3.add(BatchNormalization())
```

```

model3.add(Conv2D(64, kernel_size=3, activation=mish))
model3.add(BatchNormalization())
model3.add(Conv2D(64, kernel_size=5, strides=2, padding='same',
    ↪activation=mish))
model3.add(BatchNormalization())
model3.add(Dropout(0.4))

# 3rd Convolution Layer
model3.add(Conv2D(128, kernel_size = 4, activation=mish))
model3.add(BatchNormalization())

# Passing to a Fully Connected Layer
model3.add(Flatten())
model3.add(Dropout(0.4))

# Output Layer

# Increasing the softmax temperature
model3.add(Lambda(lambda x: x / temp))
model3.add(Dense(10, activation='softmax'))

model3.summary()

model3.compile(loss='categorical_crossentropy', optimizer=Adam(),
    ↪metrics=['accuracy'])

```

Model: "sequential_12"

Layer (type)	Output Shape	Param #
conv2d_70 (Conv2D)	(None, 26, 26, 32)	320
batch_normalization_70 (Batch Normalization)	(None, 26, 26, 32)	128
conv2d_71 (Conv2D)	(None, 24, 24, 32)	9248
batch_normalization_71 (Batch Normalization)	(None, 24, 24, 32)	128
conv2d_72 (Conv2D)	(None, 12, 12, 32)	25632
batch_normalization_72 (Batch Normalization)	(None, 12, 12, 32)	128
dropout_30 (Dropout)	(None, 12, 12, 32)	0
conv2d_73 (Conv2D)	(None, 10, 10, 64)	18496
batch_normalization_73 (Batch Normalization)	(None, 10, 10, 64)	256

conv2d_74 (Conv2D)	(None, 8, 8, 64)	36928
batch_normalization_74 (Batch Normalization)	(None, 8, 8, 64)	256
conv2d_75 (Conv2D)	(None, 4, 4, 64)	102464
batch_normalization_75 (Batch Normalization)	(None, 4, 4, 64)	256
dropout_31 (Dropout)	(None, 4, 4, 64)	0
conv2d_76 (Conv2D)	(None, 1, 1, 128)	131200
batch_normalization_76 (Batch Normalization)	(None, 1, 1, 128)	512
flatten_10 (Flatten)	(None, 128)	0
dropout_32 (Dropout)	(None, 128)	0
lambda_15 (Lambda)	(None, 128)	0
dense_10 (Dense)	(None, 10)	1290

Total params: 327,242
 Trainable params: 326,410
 Non-trainable params: 832

I've used higher early stopping patience level for this as this had so much variance.

```
[26]: checkpoint_filepath3 = 'part3/checkpoint'
model_checkpoint_callback3 = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath3,
    save_weights_only=True,
    monitor='val_loss',
    mode='min',
    save_best_only=True)

history3 = model3.fit(
    train_generator3,
    epochs=EPOCHS,
    validation_data=mnist_test_gen,
    steps_per_epoch = train_generator3.samples // BATCH_SIZE // BATCH_SIZE,
    validation_steps = len(x_test) // BATCH_SIZE,
    callbacks=[model_checkpoint_callback3, early_stopping_callback2]
)
```

Epoch 1/400

14/14 [=====] - 2s 84ms/step - loss: 2.3560 - accuracy:
0.0673 - val_loss: 2.3038 - val_accuracy: 0.1134
Epoch 2/400
14/14 [=====] - 1s 65ms/step - loss: 2.3629 - accuracy:
0.1098 - val_loss: 2.3078 - val_accuracy: 0.0983
Epoch 3/400
14/14 [=====] - 1s 66ms/step - loss: 2.3092 - accuracy:
0.1332 - val_loss: 2.3134 - val_accuracy: 0.0599
Epoch 4/400
14/14 [=====] - 1s 65ms/step - loss: 2.3516 - accuracy:
0.1037 - val_loss: 2.3211 - val_accuracy: 0.1011
Epoch 5/400
14/14 [=====] - 1s 65ms/step - loss: 2.3363 - accuracy:
0.0944 - val_loss: 2.3302 - val_accuracy: 0.0277
Epoch 6/400
14/14 [=====] - 1s 65ms/step - loss: 2.3161 - accuracy:
0.1145 - val_loss: 2.3350 - val_accuracy: 0.0829
Epoch 7/400
14/14 [=====] - 1s 65ms/step - loss: 2.3538 - accuracy:
0.0814 - val_loss: 2.3486 - val_accuracy: 0.0800
Epoch 8/400
14/14 [=====] - 1s 63ms/step - loss: 2.3229 - accuracy:
0.1191 - val_loss: 2.3587 - val_accuracy: 0.0299
Epoch 9/400
14/14 [=====] - 1s 63ms/step - loss: 2.3041 - accuracy:
0.1250 - val_loss: 2.3713 - val_accuracy: 0.0374
Epoch 10/400
14/14 [=====] - 1s 66ms/step - loss: 2.3078 - accuracy:
0.1282 - val_loss: 2.3984 - val_accuracy: 0.0127
Epoch 11/400
14/14 [=====] - 1s 65ms/step - loss: 2.3197 - accuracy:
0.1007 - val_loss: 2.4467 - val_accuracy: 0.0042
Epoch 12/400
14/14 [=====] - 1s 66ms/step - loss: 2.2983 - accuracy:
0.1205 - val_loss: 2.4650 - val_accuracy: 0.0075
Epoch 13/400
14/14 [=====] - 1s 66ms/step - loss: 2.2914 - accuracy:
0.1102 - val_loss: 2.4710 - val_accuracy: 0.0188
Epoch 14/400
14/14 [=====] - 1s 65ms/step - loss: 2.3113 - accuracy:
0.0930 - val_loss: 2.4666 - val_accuracy: 0.0178
Epoch 15/400
14/14 [=====] - 1s 64ms/step - loss: 2.3101 - accuracy:
0.1200 - val_loss: 2.4746 - val_accuracy: 0.0163
Epoch 16/400
14/14 [=====] - 1s 65ms/step - loss: 2.2973 - accuracy:
0.1303 - val_loss: 2.4948 - val_accuracy: 0.0193
Epoch 17/400

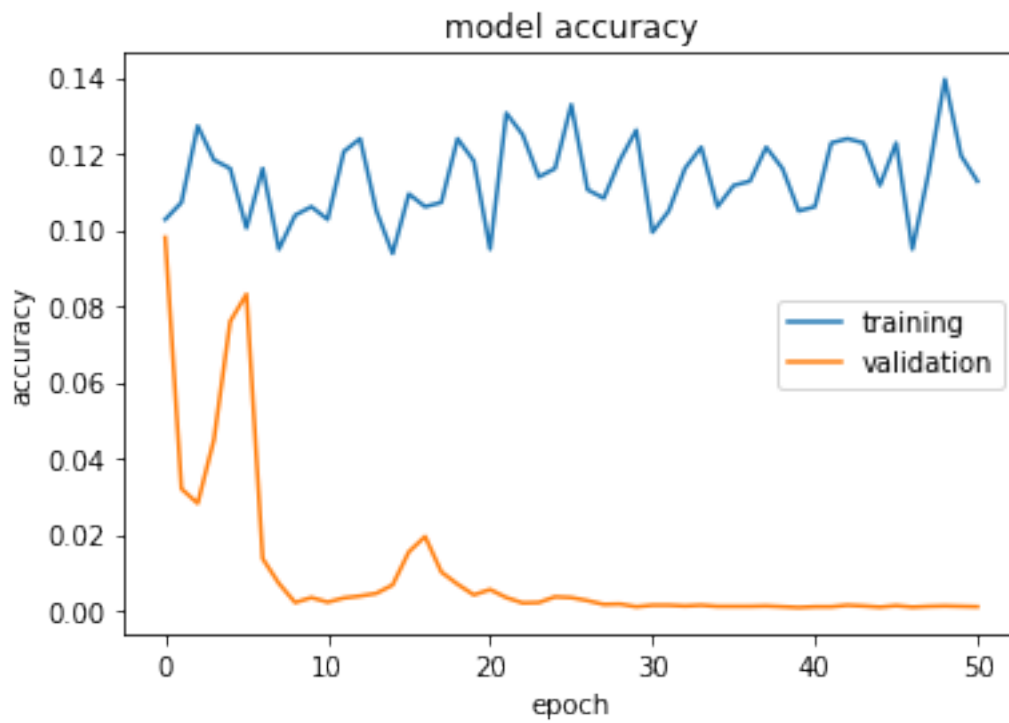
14/14 [=====] - 1s 66ms/step - loss: 2.2860 - accuracy:
0.1076 - val_loss: 2.5148 - val_accuracy: 0.0414
Epoch 18/400
14/14 [=====] - 1s 65ms/step - loss: 2.2952 - accuracy:
0.1066 - val_loss: 2.5149 - val_accuracy: 0.0193
Epoch 19/400
14/14 [=====] - 1s 65ms/step - loss: 2.2688 - accuracy:
0.1178 - val_loss: 2.5237 - val_accuracy: 0.0334
Epoch 20/400
14/14 [=====] - 1s 64ms/step - loss: 2.2893 - accuracy:
0.0942 - val_loss: 2.5546 - val_accuracy: 0.0259
Epoch 21/400
14/14 [=====] - 1s 66ms/step - loss: 2.3038 - accuracy:
0.0960 - val_loss: 2.5874 - val_accuracy: 0.0163
Epoch 22/400
14/14 [=====] - 1s 65ms/step - loss: 2.2873 - accuracy:
0.1244 - val_loss: 2.6035 - val_accuracy: 0.0159
Epoch 23/400
14/14 [=====] - 1s 65ms/step - loss: 2.3042 - accuracy:
0.1054 - val_loss: 2.6034 - val_accuracy: 0.0459
Epoch 24/400
14/14 [=====] - 1s 65ms/step - loss: 2.2683 - accuracy:
0.0998 - val_loss: 2.6401 - val_accuracy: 0.0337
Epoch 25/400
14/14 [=====] - 1s 65ms/step - loss: 2.2844 - accuracy:
0.1200 - val_loss: 2.7006 - val_accuracy: 0.0149
Epoch 26/400
14/14 [=====] - 1s 65ms/step - loss: 2.2652 - accuracy:
0.1129 - val_loss: 2.7484 - val_accuracy: 0.0149
Epoch 27/400
14/14 [=====] - 1s 66ms/step - loss: 2.2800 - accuracy:
0.1070 - val_loss: 2.7915 - val_accuracy: 0.0093
Epoch 28/400
14/14 [=====] - 1s 65ms/step - loss: 2.2683 - accuracy:
0.1406 - val_loss: 2.8479 - val_accuracy: 0.0037
Epoch 29/400
14/14 [=====] - 1s 65ms/step - loss: 2.2724 - accuracy:
0.1091 - val_loss: 2.8790 - val_accuracy: 0.0033
Epoch 30/400
14/14 [=====] - 1s 66ms/step - loss: 2.2847 - accuracy:
0.1065 - val_loss: 2.9107 - val_accuracy: 0.0035
Epoch 31/400
14/14 [=====] - 1s 66ms/step - loss: 2.2935 - accuracy:
0.1025 - val_loss: 2.8962 - val_accuracy: 0.0026
Epoch 32/400
14/14 [=====] - 1s 66ms/step - loss: 2.2958 - accuracy:
0.1069 - val_loss: 2.8861 - val_accuracy: 0.0019
Epoch 33/400

14/14 [=====] - 1s 64ms/step - loss: 2.2440 - accuracy: 0.1436 - val_loss: 2.9449 - val_accuracy: 0.0017
Epoch 34/400
14/14 [=====] - 1s 64ms/step - loss: 2.2585 - accuracy: 0.1440 - val_loss: 3.0201 - val_accuracy: 0.0010
Epoch 35/400
14/14 [=====] - 1s 64ms/step - loss: 2.2694 - accuracy: 0.1164 - val_loss: 3.0752 - val_accuracy: 0.0012
Epoch 36/400
14/14 [=====] - 1s 64ms/step - loss: 2.2912 - accuracy: 0.1181 - val_loss: 3.0807 - val_accuracy: 0.0012
Epoch 37/400
14/14 [=====] - 1s 65ms/step - loss: 2.2644 - accuracy: 0.1260 - val_loss: 3.1123 - val_accuracy: 9.0144e-04
Epoch 38/400
14/14 [=====] - 1s 65ms/step - loss: 2.2493 - accuracy: 0.1565 - val_loss: 3.1913 - val_accuracy: 9.0144e-04
Epoch 39/400
14/14 [=====] - 1s 63ms/step - loss: 2.2711 - accuracy: 0.1216 - val_loss: 3.2134 - val_accuracy: 0.0012
Epoch 40/400
14/14 [=====] - 1s 64ms/step - loss: 2.2919 - accuracy: 0.1140 - val_loss: 3.2917 - val_accuracy: 0.0012
Epoch 41/400
14/14 [=====] - 1s 65ms/step - loss: 2.2788 - accuracy: 0.1079 - val_loss: 3.3463 - val_accuracy: 0.0017
Epoch 42/400
14/14 [=====] - 1s 66ms/step - loss: 2.2831 - accuracy: 0.1148 - val_loss: 3.3265 - val_accuracy: 0.0014
Epoch 43/400
14/14 [=====] - 1s 64ms/step - loss: 2.2798 - accuracy: 0.0903 - val_loss: 3.3340 - val_accuracy: 6.0096e-04
Epoch 44/400
14/14 [=====] - 1s 65ms/step - loss: 2.2873 - accuracy: 0.1235 - val_loss: 3.3501 - val_accuracy: 5.0080e-04
Epoch 45/400
14/14 [=====] - 1s 63ms/step - loss: 2.2556 - accuracy: 0.1139 - val_loss: 3.3654 - val_accuracy: 6.0096e-04
Epoch 46/400
14/14 [=====] - 1s 65ms/step - loss: 2.2759 - accuracy: 0.1083 - val_loss: 3.3459 - val_accuracy: 7.0112e-04
Epoch 47/400
14/14 [=====] - 1s 66ms/step - loss: 2.2653 - accuracy: 0.1307 - val_loss: 3.3710 - val_accuracy: 8.0128e-04
Epoch 48/400
14/14 [=====] - 1s 66ms/step - loss: 2.2516 - accuracy: 0.1418 - val_loss: 3.4167 - val_accuracy: 5.0080e-04
Epoch 49/400

```
14/14 [=====] - 1s 63ms/step - loss: 2.2922 - accuracy:
0.0989 - val_loss: 3.4573 - val_accuracy: 8.0128e-04
Epoch 50/400
14/14 [=====] - 1s 65ms/step - loss: 2.2671 - accuracy:
0.1089 - val_loss: 3.4687 - val_accuracy: 7.0112e-04
Epoch 51/400
14/14 [=====] - 1s 68ms/step - loss: 2.2688 - accuracy:
0.1140 - val_loss: 3.4921 - val_accuracy: 7.0112e-04
Restoring model weights from the end of the best epoch.
Epoch 00051: early stopping
```

```
[ ]: plt.figure(1)
plt.plot(history3.history['accuracy'])
plt.plot(history3.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()

plt.figure(2)
plt.plot(history3.history['loss'])
plt.plot(history3.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```



Using the pretrained model

```
[22]: model3 = Sequential()

# Lambda Layer for adding Padding
model3.add(Lambda(lambda image: tf.image.resize_with_crop_or_pad(
    image, 28, 28), input_shape=(*IMAGE_SIZE, 1)))

# 1st Convolution Layer
model3.add(Conv2D(32, input_shape=(*IMAGE_SIZE, 1), kernel_size=3,
    ↪activation=mish))
model3.add(BatchNormalization())
model3.add(Conv2D(32, kernel_size=3, activation=mish))
model3.add(BatchNormalization())
model3.add(Conv2D(32, kernel_size=5, strides=2, padding='same',
    ↪activation=mish))
model3.add(BatchNormalization())
model3.add(Dropout(0.4))

# 2nd Convolution Layer
model3.add(Conv2D(64, kernel_size=3, activation=mish))
model3.add(BatchNormalization())
model3.add(Conv2D(64, kernel_size=3, activation=mish))
model3.add(BatchNormalization())
model3.add(Conv2D(64, kernel_size=5, strides=2, padding='same',
    ↪activation=mish))
model3.add(BatchNormalization())
model3.add(Dropout(0.4))

# 3rd Convolution Layer
model3.add(Conv2D(128, kernel_size = 4, activation=mish))
model3.add(BatchNormalization())

# Passing to a Fully Connected Layer
model3.add(Flatten())
model3.add(Dropout(0.4))

# Output Layer

# Increasing the softmax temperature
model3.add(Lambda(lambda x: x / temp))
model3.add(Dense(10, activation='softmax'))

model3.summary()

model3.load_weights('part2_pretrained2/checkpoint')
```

```
model3.compile(loss='categorical_crossentropy', optimizer=Adam(),  
↳metrics=['accuracy'])
```

```
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.iter  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.beta_1  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.beta_2  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.decay  
WARNING:tensorflow:Unresolved object in checkpoint:  
(root).optimizer.learning_rate  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-0.kernel  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-0.bias  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-1.gamma  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-1.beta  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-2.kernel  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-2.bias  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-3.gamma  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-3.beta  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-4.kernel  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-4.bias  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-5.gamma  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-5.beta  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-6.kernel  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-6.bias  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-7.gamma  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-7.beta  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-8.kernel  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-8.bias  
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'  
for (root).layer_with_weights-9.gamma
```



```

WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-6.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-7.gamma
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-7.beta
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-8.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-8.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-9.gamma
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-9.beta
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-10.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-10.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-11.gamma
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-11.beta
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-12.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-12.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-13.gamma
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-13.beta
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-14.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-14.bias
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.restore
or tf.keras.Model.load_weights) but not all checkpointed values were used. See
above for specific issues. Use expect_partial() on the load status object, e.g.
tf.train.Checkpoint.restore(...).expect_partial(), to silence these warnings, or
use assert_consumed() to make the check explicit. See
https://www.tensorflow.org/guide/checkpoint#loading\_mechanics for details.
Model: "sequential_11"

```

Layer (type)	Output Shape	Param #
lambda_13 (Lambda)	(None, 28, 28, 1)	0
conv2d_63 (Conv2D)	(None, 26, 26, 32)	320

batch_normalization_63 (Batch Normalization)	(None, 26, 26, 32)	128
conv2d_64 (Conv2D)	(None, 24, 24, 32)	9248
batch_normalization_64 (Batch Normalization)	(None, 24, 24, 32)	128
conv2d_65 (Conv2D)	(None, 12, 12, 32)	25632
batch_normalization_65 (Batch Normalization)	(None, 12, 12, 32)	128
dropout_27 (Dropout)	(None, 12, 12, 32)	0
conv2d_66 (Conv2D)	(None, 10, 10, 64)	18496
batch_normalization_66 (Batch Normalization)	(None, 10, 10, 64)	256
conv2d_67 (Conv2D)	(None, 8, 8, 64)	36928
batch_normalization_67 (Batch Normalization)	(None, 8, 8, 64)	256
conv2d_68 (Conv2D)	(None, 4, 4, 64)	102464
batch_normalization_68 (Batch Normalization)	(None, 4, 4, 64)	256
dropout_28 (Dropout)	(None, 4, 4, 64)	0
conv2d_69 (Conv2D)	(None, 1, 1, 128)	131200
batch_normalization_69 (Batch Normalization)	(None, 1, 1, 128)	512
flatten_9 (Flatten)	(None, 128)	0
dropout_29 (Dropout)	(None, 128)	0
lambda_14 (Lambda)	(None, 128)	0
dense_9 (Dense)	(None, 10)	1290

=====
 Total params: 327,242
 Trainable params: 326,410
 Non-trainable params: 832
 =====

```
[23]: checkpoint_filepath3un = 'part3_un/checkpoint'
      model_checkpoint_callback3un = tf.keras.callbacks.ModelCheckpoint(
          filepath=checkpoint_filepath3un,
          save_weights_only=True,
```

```

        monitor='val_loss',
        mode='min',
        save_best_only=True)

history3un = model3.fit(
    train_generator3,
    epochs=EPOCHS,
    validation_data=mnist_test_gen,
    steps_per_epoch = train_generator3.samples // BATCH_SIZE // BATCH_SIZE,
    validation_steps = len(x_test) // BATCH_SIZE,
    callbacks=[model_checkpoint_callback3un, early_stopping_callback2]
)

```

Epoch 1/400

14/14 [=====] - 32s 86ms/step - loss: 2.4406 - accuracy: 0.1039 - val_loss: 2.3019 - val_accuracy: 0.1136

Epoch 2/400

14/14 [=====] - 1s 65ms/step - loss: 2.4084 - accuracy: 0.1114 - val_loss: 2.3183 - val_accuracy: 0.1137

Epoch 3/400

14/14 [=====] - 1s 64ms/step - loss: 2.3718 - accuracy: 0.0867 - val_loss: 2.3375 - val_accuracy: 0.0421

Epoch 4/400

14/14 [=====] - 1s 65ms/step - loss: 2.3478 - accuracy: 0.1095 - val_loss: 2.3601 - val_accuracy: 0.0230

Epoch 5/400

14/14 [=====] - 1s 68ms/step - loss: 2.3320 - accuracy: 0.1133 - val_loss: 2.3728 - val_accuracy: 0.0239

Epoch 6/400

14/14 [=====] - 1s 68ms/step - loss: 2.3150 - accuracy: 0.1258 - val_loss: 2.3833 - val_accuracy: 0.0460

Epoch 7/400

14/14 [=====] - 1s 65ms/step - loss: 2.3344 - accuracy: 0.1201 - val_loss: 2.3946 - val_accuracy: 0.0153

Epoch 8/400

14/14 [=====] - 1s 65ms/step - loss: 2.3208 - accuracy: 0.0992 - val_loss: 2.4039 - val_accuracy: 0.0146

Epoch 9/400

14/14 [=====] - 1s 67ms/step - loss: 2.3020 - accuracy: 0.1062 - val_loss: 2.4324 - val_accuracy: 0.0389

Epoch 10/400

14/14 [=====] - 1s 66ms/step - loss: 2.3130 - accuracy: 0.1032 - val_loss: 2.4525 - val_accuracy: 0.0406

Epoch 11/400

14/14 [=====] - 1s 66ms/step - loss: 2.3078 - accuracy: 0.1355 - val_loss: 2.4655 - val_accuracy: 0.0409

Epoch 12/400

14/14 [=====] - 1s 67ms/step - loss: 2.2980 - accuracy:
0.0997 - val_loss: 2.4796 - val_accuracy: 0.0287
Epoch 13/400
14/14 [=====] - 1s 66ms/step - loss: 2.2810 - accuracy:
0.1304 - val_loss: 2.4900 - val_accuracy: 0.0233
Epoch 14/400
14/14 [=====] - 1s 65ms/step - loss: 2.2815 - accuracy:
0.0958 - val_loss: 2.5059 - val_accuracy: 0.0117
Epoch 15/400
14/14 [=====] - 1s 66ms/step - loss: 2.2735 - accuracy:
0.1081 - val_loss: 2.5084 - val_accuracy: 0.0365
Epoch 16/400
14/14 [=====] - 1s 68ms/step - loss: 2.3062 - accuracy:
0.1062 - val_loss: 2.5373 - val_accuracy: 0.0548
Epoch 17/400
14/14 [=====] - 1s 66ms/step - loss: 2.2913 - accuracy:
0.1321 - val_loss: 2.5145 - val_accuracy: 0.0383
Epoch 18/400
14/14 [=====] - 1s 65ms/step - loss: 2.2897 - accuracy:
0.1007 - val_loss: 2.5045 - val_accuracy: 0.0147
Epoch 19/400
14/14 [=====] - 1s 65ms/step - loss: 2.2944 - accuracy:
0.0994 - val_loss: 2.4773 - val_accuracy: 0.0047
Epoch 20/400
14/14 [=====] - 1s 66ms/step - loss: 2.2799 - accuracy:
0.1305 - val_loss: 2.4635 - val_accuracy: 0.0072
Epoch 21/400
14/14 [=====] - 1s 64ms/step - loss: 2.2692 - accuracy:
0.1524 - val_loss: 2.4767 - val_accuracy: 0.0145
Epoch 22/400
14/14 [=====] - 1s 65ms/step - loss: 2.2738 - accuracy:
0.1329 - val_loss: 2.5047 - val_accuracy: 0.0252
Epoch 23/400
14/14 [=====] - 1s 66ms/step - loss: 2.2971 - accuracy:
0.1013 - val_loss: 2.5380 - val_accuracy: 0.0227
Epoch 24/400
14/14 [=====] - 1s 66ms/step - loss: 2.2871 - accuracy:
0.1093 - val_loss: 2.5883 - val_accuracy: 0.0066
Epoch 25/400
14/14 [=====] - 1s 64ms/step - loss: 2.2785 - accuracy:
0.1376 - val_loss: 2.6444 - val_accuracy: 0.0034
Epoch 26/400
14/14 [=====] - 1s 66ms/step - loss: 2.2682 - accuracy:
0.1249 - val_loss: 2.6679 - val_accuracy: 0.0076
Epoch 27/400
14/14 [=====] - 1s 64ms/step - loss: 2.2823 - accuracy:
0.1293 - val_loss: 2.7002 - val_accuracy: 0.0069
Epoch 28/400

14/14 [=====] - 1s 65ms/step - loss: 2.3061 - accuracy:
 0.0920 - val_loss: 2.7721 - val_accuracy: 0.0052
 Epoch 29/400
 14/14 [=====] - 1s 65ms/step - loss: 2.2853 - accuracy:
 0.1027 - val_loss: 2.8458 - val_accuracy: 0.0038
 Epoch 30/400
 14/14 [=====] - 1s 67ms/step - loss: 2.2862 - accuracy:
 0.1180 - val_loss: 2.8777 - val_accuracy: 0.0021
 Epoch 31/400
 14/14 [=====] - 1s 65ms/step - loss: 2.3013 - accuracy:
 0.0922 - val_loss: 2.8975 - val_accuracy: 0.0015
 Epoch 32/400
 14/14 [=====] - 1s 65ms/step - loss: 2.2633 - accuracy:
 0.1137 - val_loss: 2.9175 - val_accuracy: 0.0014
 Epoch 33/400
 14/14 [=====] - 1s 65ms/step - loss: 2.2684 - accuracy:
 0.1376 - val_loss: 2.9901 - val_accuracy: 0.0022
 Epoch 34/400
 14/14 [=====] - 1s 66ms/step - loss: 2.2626 - accuracy:
 0.1175 - val_loss: 3.0472 - val_accuracy: 0.0017
 Epoch 35/400
 14/14 [=====] - 1s 66ms/step - loss: 2.2700 - accuracy:
 0.1299 - val_loss: 3.1571 - val_accuracy: 0.0013
 Epoch 36/400
 14/14 [=====] - 1s 66ms/step - loss: 2.2644 - accuracy:
 0.1170 - val_loss: 3.2223 - val_accuracy: 8.0128e-04
 Epoch 37/400
 14/14 [=====] - 1s 65ms/step - loss: 2.2730 - accuracy:
 0.1162 - val_loss: 3.2196 - val_accuracy: 0.0013
 Epoch 38/400
 14/14 [=====] - 1s 65ms/step - loss: 2.2711 - accuracy:
 0.0958 - val_loss: 3.2479 - val_accuracy: 3.0048e-04
 Epoch 39/400
 14/14 [=====] - 1s 65ms/step - loss: 2.2524 - accuracy:
 0.1328 - val_loss: 3.2717 - val_accuracy: 8.0128e-04
 Epoch 40/400
 14/14 [=====] - 1s 63ms/step - loss: 2.2900 - accuracy:
 0.1266 - val_loss: 3.3215 - val_accuracy: 8.0128e-04
 Epoch 41/400
 14/14 [=====] - 1s 66ms/step - loss: 2.2842 - accuracy:
 0.1065 - val_loss: 3.3232 - val_accuracy: 0.0011
 Epoch 42/400
 14/14 [=====] - 1s 66ms/step - loss: 2.2621 - accuracy:
 0.1510 - val_loss: 3.2961 - val_accuracy: 0.0015
 Epoch 43/400
 14/14 [=====] - 1s 65ms/step - loss: 2.2697 - accuracy:
 0.1044 - val_loss: 3.3119 - val_accuracy: 0.0013
 Epoch 44/400

```

14/14 [=====] - 1s 66ms/step - loss: 2.2752 - accuracy:
0.1215 - val_loss: 3.3611 - val_accuracy: 0.0017
Epoch 45/400
14/14 [=====] - 1s 63ms/step - loss: 2.2518 - accuracy:
0.1388 - val_loss: 3.3865 - val_accuracy: 0.0013
Epoch 46/400
14/14 [=====] - 1s 65ms/step - loss: 2.2691 - accuracy:
0.0855 - val_loss: 3.3896 - val_accuracy: 0.0012
Epoch 47/400
14/14 [=====] - 1s 65ms/step - loss: 2.2696 - accuracy:
0.1072 - val_loss: 3.4085 - val_accuracy: 9.0144e-04
Epoch 48/400
14/14 [=====] - 1s 64ms/step - loss: 2.2620 - accuracy:
0.1289 - val_loss: 3.4581 - val_accuracy: 8.0128e-04
Epoch 49/400
14/14 [=====] - 1s 64ms/step - loss: 2.2829 - accuracy:
0.1055 - val_loss: 3.4740 - val_accuracy: 0.0010
Epoch 50/400
14/14 [=====] - 1s 65ms/step - loss: 2.2791 - accuracy:
0.1036 - val_loss: 3.4407 - val_accuracy: 0.0012
Epoch 51/400
14/14 [=====] - 1s 65ms/step - loss: 2.2756 - accuracy:
0.0929 - val_loss: 3.4505 - val_accuracy: 0.0019
Restoring model weights from the end of the best epoch.
Epoch 00051: early stopping

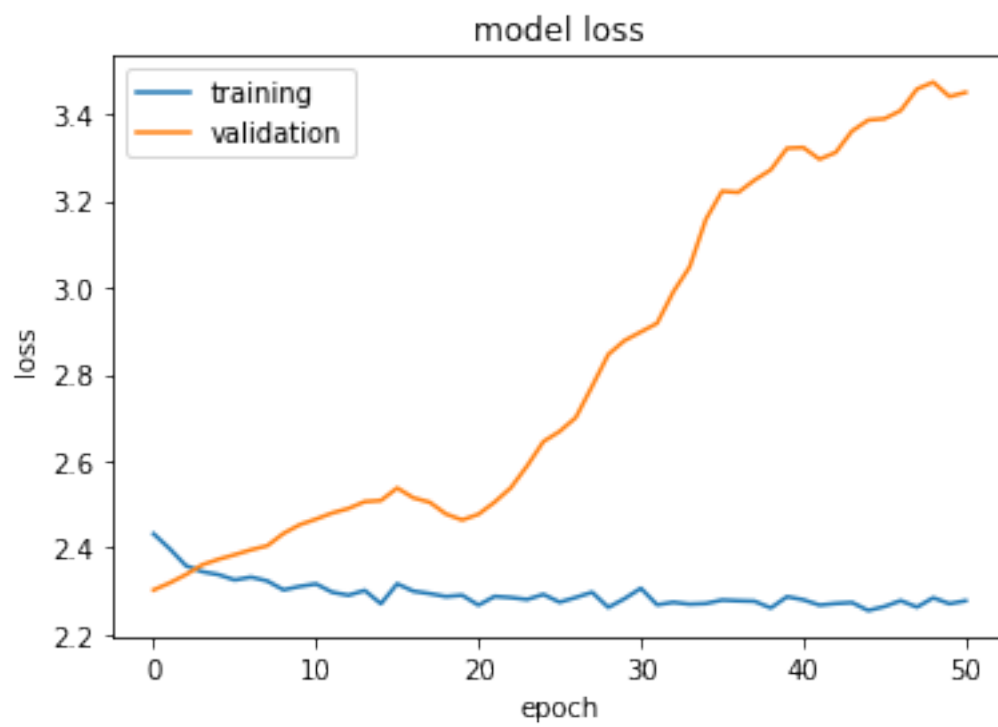
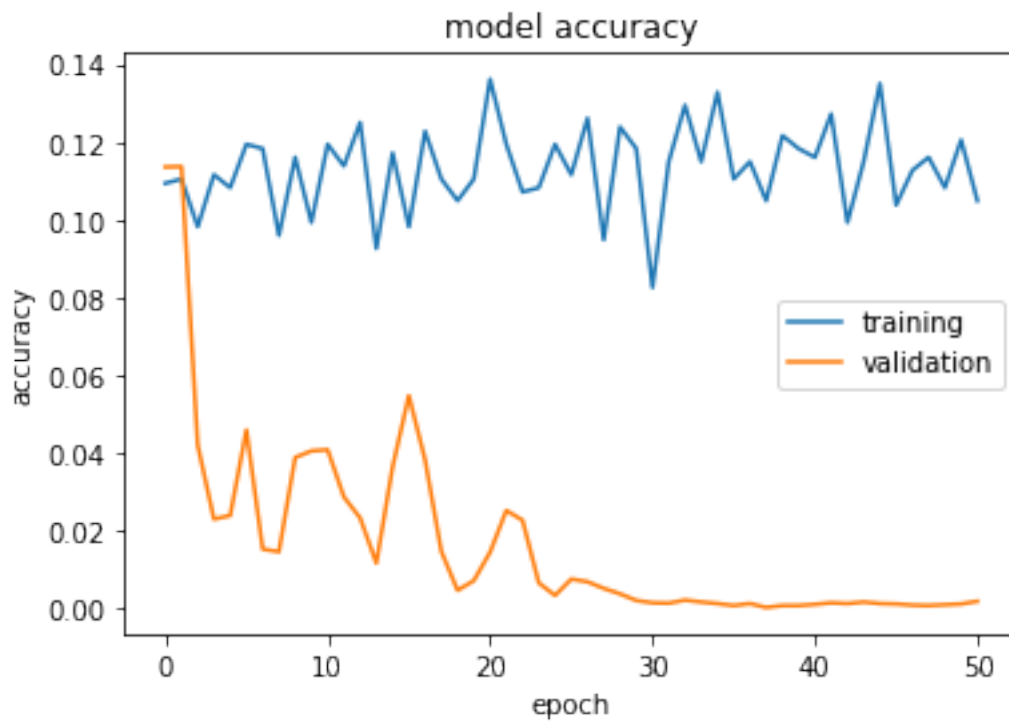
```

```

[24]: plt.figure(1)
plt.plot(history3un.history['accuracy'])
plt.plot(history3un.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()

plt.figure(2)
plt.plot(history3un.history['loss'])
plt.plot(history3un.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()

```



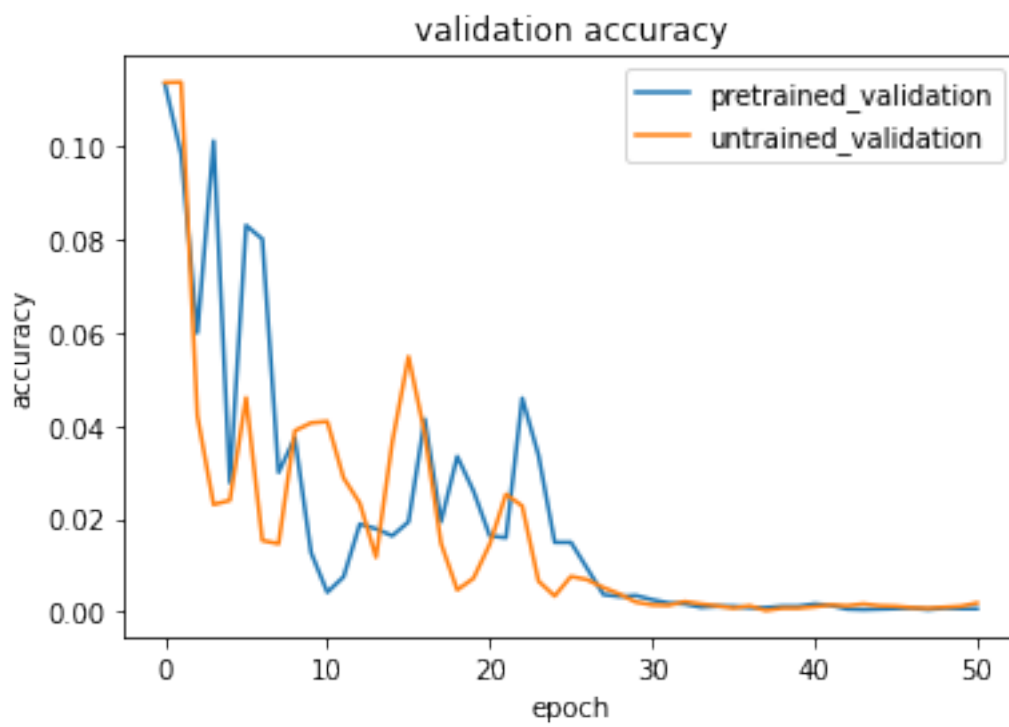
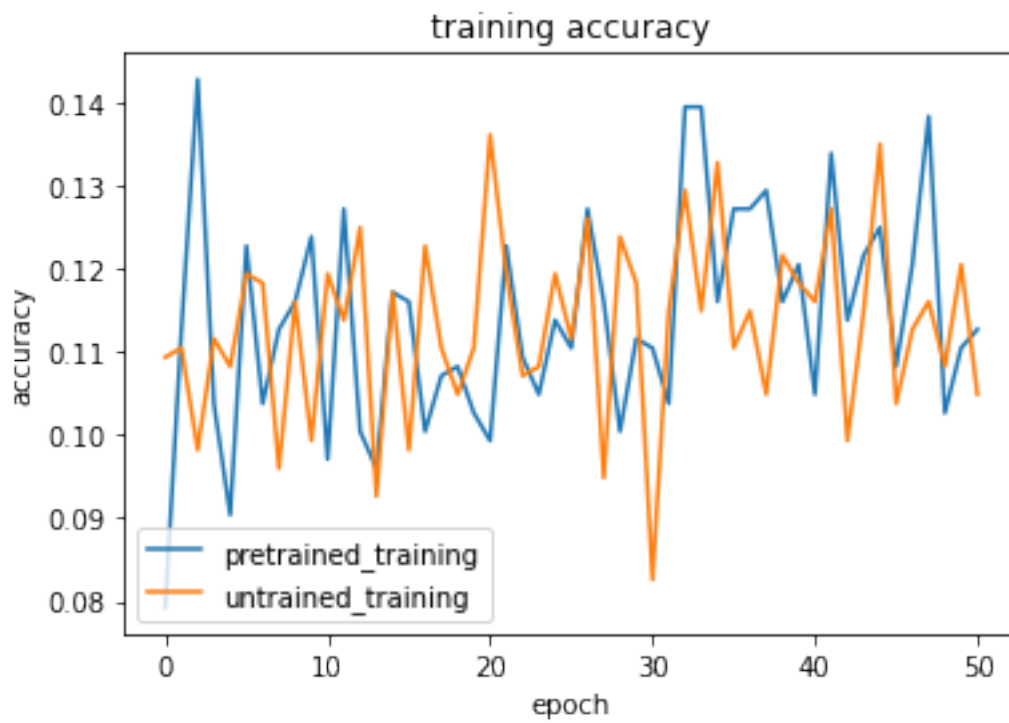
2.3 Observation and results

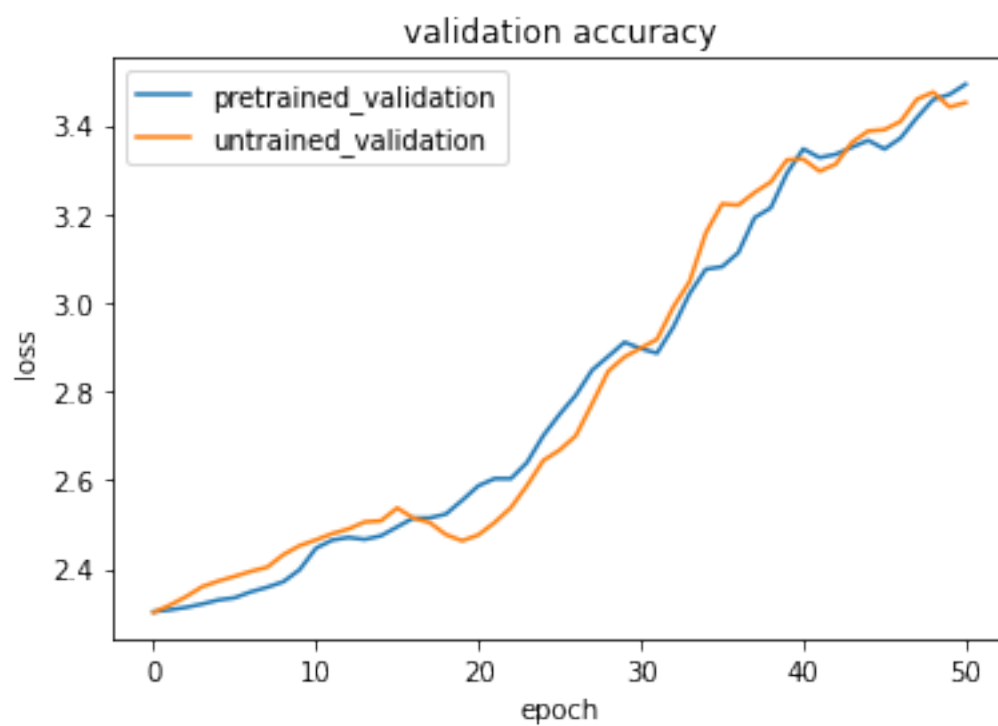
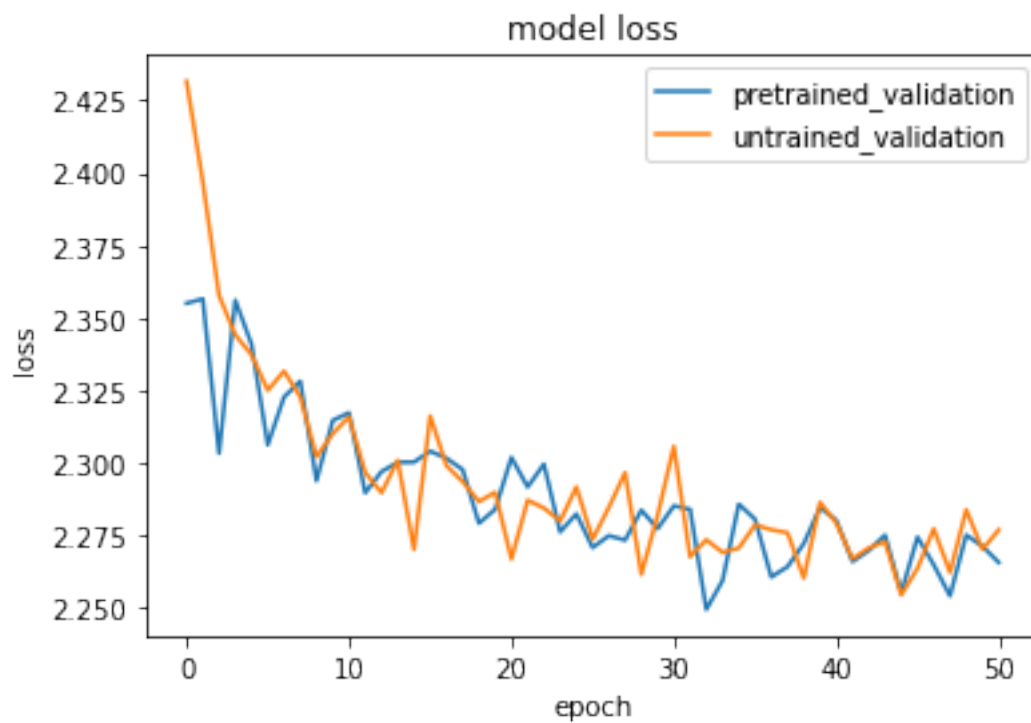
```
[28]: plt.figure(1)
plt.plot(history3.history['accuracy'])
plt.plot(history3un.history['accuracy'])
plt.title('training accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['pretrained_training', 'untrained_training'], loc='best')
plt.show()

plt.figure(2)
plt.plot(history3.history['val_accuracy'])
plt.plot(history3un.history['val_accuracy'])
plt.title('validation accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['pretrained_validation', 'untrained_validation'], loc='best')
plt.show()

plt.figure(3)
plt.plot(history3.history['loss'])
plt.plot(history3un.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['pretrained_validation', 'untrained_validation'], loc='best')
plt.show()

plt.figure(4)
plt.plot(history3.history['val_loss'])
plt.plot(history3un.history['val_loss'])
plt.title('validation accuracy')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['pretrained_validation', 'untrained_validation'], loc='best')
plt.show()
```



Since the pre-trained network was trained on the correct labels and the labels in dataset of this task are mixed, it's not giving any correct labels, and is similar in performance to the randomly initialized model.

However, the pretrained model has lower validation loss and higher validation accuracy for most part, but overall, they are still very close.

These both models does not perform any good. This is because of the data provided, as we have seen pretty good results with this model.

This dataset comes from the MNIST distribution, but has been randomly shuffled, so the labels no more correspond to the correct image. Since this was tested on the correctly labelled MNIST dataset, it performed very badly as it had learned the wrong weights.

This is a classic example of how we should always check and verify that the data that we have is cleaned, properly labelled and unbiased as even good models give very bad and biased predictions if the data is incorrect.

[]: