

# midastask2part1

April 10, 2021

Author: Pushkar Patel

## 1 Task 2 - Part 1

Imports

```
[10]: import shutil
      from PIL import Image
      import numpy as np
      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
```

Extracting the images

```
[2]: shutil.unpack_archive('trainPart1.zip', '../input/trainpart1zip')
```

### 1.1 Inspecting the image

Browsing through the dataset, we can see that there are a total of 62 classes - 10 numbers from 0 to 9, 26 lowercase alphabets and 26 uppercase alphabets, having 40 examples each.

Inspecting the image to view its dimensions and colour channels

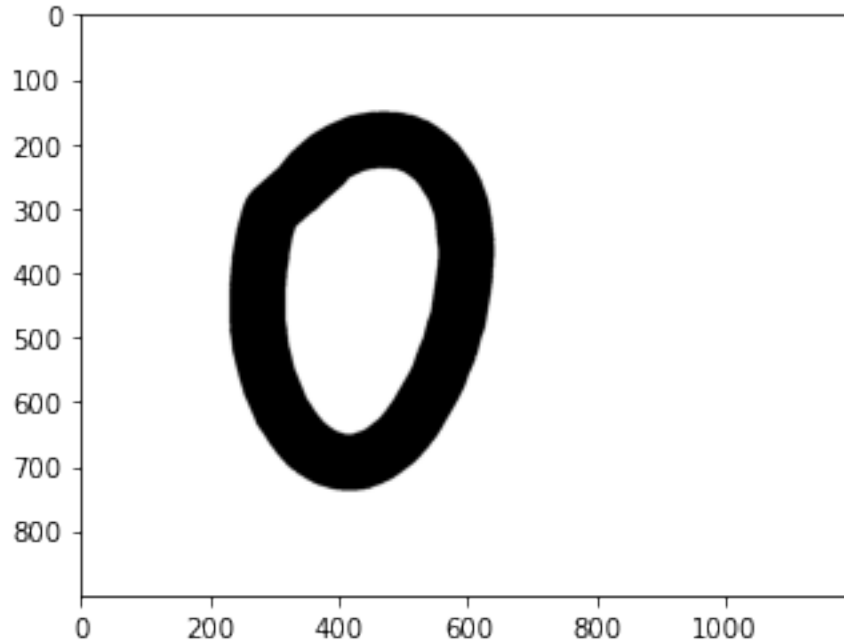
```
[ ]: image = Image.open('../input/trainpart1zip/train/Sample001/img001-001.png')
      np_image = np.array(image)
      print(np_image.shape)
      print(image.mode)
```

```
imshow(image)
```

```
(900, 1200, 3)
```

```
RGB
```

```
[ ]: <matplotlib.image.AxesImage at 0x7f6d8e70a2b0>
```



The image is of dimension 900x1200 with three colour channels. Looking at the images in the directory, I found that all the images are black and white and contain only handwritten digits or alphabets. We can convert them to single grayscale colour channel to reduce computations, improve speed and make the architecture less complicated.

## 1.2 Preprocessing the images to convert test and validation input and labels

NOTE: Initially, I tried to build the network on the full image dimension of 900x1200, but that just overloaded the memory with too many parameters. Scaling down the dimensions, I found that reducing the image by 20x i.e. image of dimension 45x60 has comparatively smaller number of parameters to train and the images are still recognizable from each other.

I'll use ImageDataGenerator from Keras to preprocess and split the training images into train and validation sets.

I've normalized all pixel values to be in the range of 0 to 1 for the data to have similar range.

I split the training and validation sets in 80:20 ratio.

```
[3]: train_datagen1 = ImageDataGenerator(rescale=1./255, validation_split=0.2)
```

I create the generator object which would generate the training and validation sets. It takes the input from the images folder. I'm reducing the size of the images by 20x while taking the input, and changing the colour channel to grayscale. Each set is of batch size 64. I chose this as it's a good enough batch size for this size of dataset. The class labels are categorical and are one-hot encoded for all of the 62 classes (10 numbers + 26 lowercase alphabets + 26 uppercase alphabets).

Variables that would be used globally

```
[4]: BATCH_SIZE = 64
      IMAGE_SIZE = (45, 60)
      EPOCHS = 400
```

```
[5]: train_generator1 = train_datagen1.flow_from_directory(
      '../input/trainpart1zip/train',
      target_size=IMAGE_SIZE,
      batch_size=BATCH_SIZE,
      class_mode='categorical',
      color_mode='grayscale',
      subset='training',
      seed=42,
      shuffle=True)
```

Found 1984 images belonging to 62 classes.

```
[6]: validation_generator1 = train_datagen1.flow_from_directory(
      '../input/trainpart1zip/train',
      target_size=tf.squeeze(IMAGE_SIZE),
      batch_size=BATCH_SIZE,
      class_mode='categorical',
      color_mode='grayscale',
      subset='validation',
      seed=42,
      shuffle=True)
```

Found 496 images belonging to 62 classes.

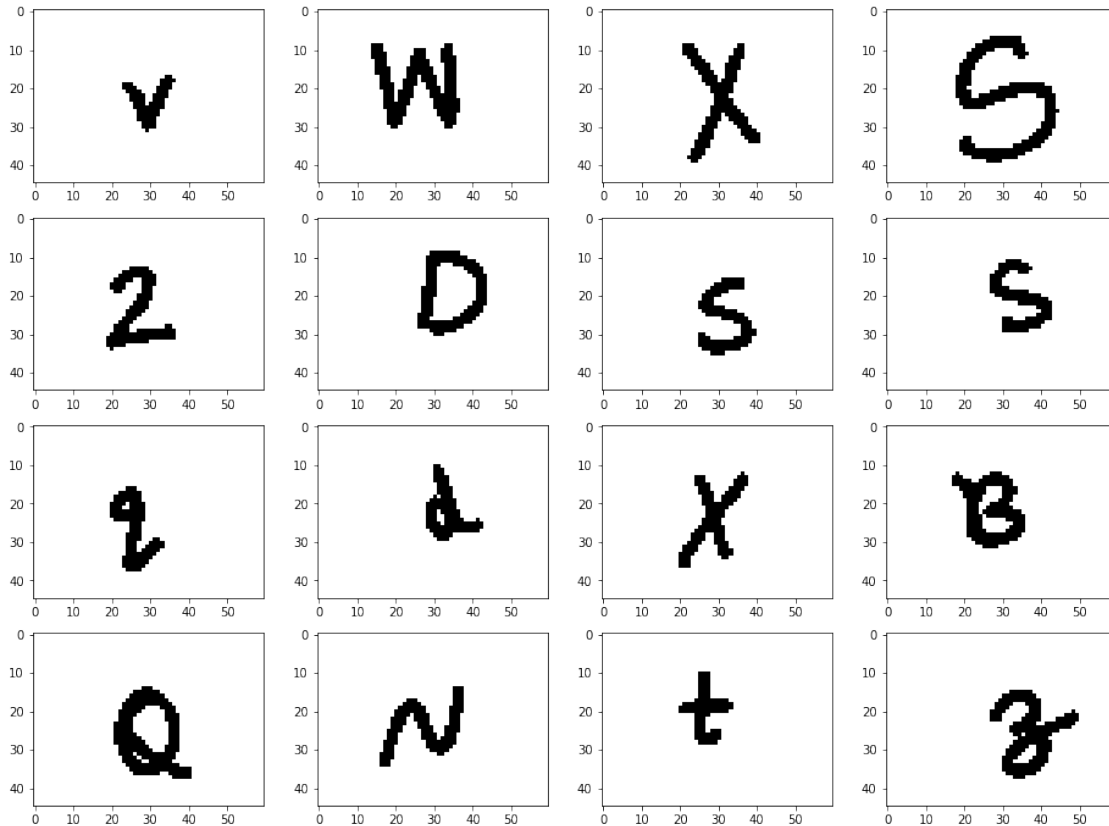
The ImageDataGenerator class has automatically detected the 62 classes and has one-hot encoded them accordingly.

Viewing the generated samples

```
[ ]: X_train_batch0, y_train_batch0 = train_generator1.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()
```

(64, 45, 60, 1) (64, 62)

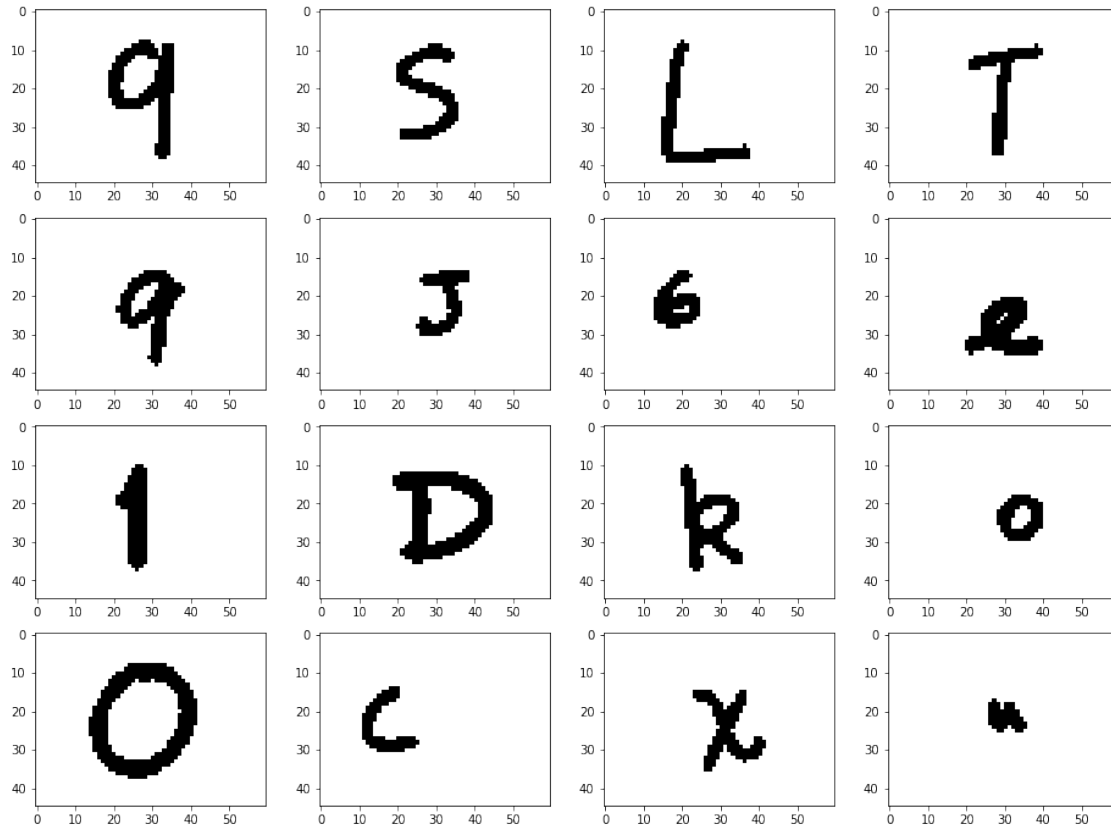
```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```



```
[ ]: X_validation_batch0, y_validation_batch0 = validation_generator1.next()
print(X_validation_batch0.shape, y_validation_batch0.shape)
print(y_validation_batch0[0])
plt.figure(figsize=(16,12))
for i in range(1, 17):
    plt.subplot(4,4,i)
    imshow(tf.squeeze(X_validation_batch0[i]), cmap='gray')
plt.show()
```

(64, 45, 60, 1) (64, 62)

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



## 1.3 Building the model

### 1.3.1 Experiment 1: Building the first model inspired from LeNet

I quickly build a first model, which inspired by the original LeNet, with slight modifications, to check how it performs and will then tune the hyperparameter accordingly. I also use dropouts with a probability of 0.4 for each Fully Connected Layer.

```
[ ]: model1 = Sequential()

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

# 2nd Convolution Layer
model1.add(Conv2D(16, kernel_size=(5,5), activation='relu'))
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(256, activation='relu'))
model1.add(BatchNormalization())
model1.add(Dropout(0.4))

# 2nd Fully Connected Layer
model1.add(Dense(128, activation='relu'))
model1.add(BatchNormalization())
model1.add(Dropout(0.4))

# Output Layer
model1.add(Dense(62, activation='softmax'))

```

```
[ ]: model1.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 45, 60, 6)	156
batch_normalization (Batch Normalization)	(None, 45, 60, 6)	24
max_pooling2d (MaxPooling2D)	(None, 22, 30, 6)	0
conv2d_1 (Conv2D)	(None, 18, 26, 16)	2416
batch_normalization_1 (Batch Normalization)	(None, 18, 26, 16)	64
max_pooling2d_1 (MaxPooling2D)	(None, 9, 13, 16)	0
flatten (Flatten)	(None, 1872)	0
dense (Dense)	(None, 256)	479488
batch_normalization_2 (Batch Normalization)	(None, 256)	1024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32896
batch_normalization_3 (Batch Normalization)	(None, 128)	512
dropout_1 (Dropout)	(None, 128)	0

```

-----
dense_2 (Dense)                (None, 62)                7998
=====
Total params: 524,578
Trainable params: 523,766
Non-trainable params: 812
-----

```

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

Using Early Stopping

```
[7]: early_stopping_callback = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    mode='min',
    patience=20,
    restore_best_weights=True,
    verbose=1)
```

Saving the checkpoint

```
[ ]: checkpoint_filepath1 = 'exp1/checkpoint'
model_checkpoint_callback1 = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath1,
    save_weights_only=True,
    monitor='val_loss',
    mode='min',
    save_best_only=True)
```

I train the model for 400 epochs and use early stopping with a patience level of 20 epochs in order to prevent model from overfitting and save the best weights of the mode.

```
[ ]: history1 = model1.fit(
    train_generator1,
    epochs=EPOCHS,
    validation_data=validation_generator1,
    steps_per_epoch = train_generator1.samples // BATCH_SIZE,
    validation_steps = validation_generator1.samples // BATCH_SIZE,
    callbacks=[model_checkpoint_callback1, early_stopping_callback]
)
```

Epoch 1/400

```
31/31 [=====] - 43s 1s/step - loss: 4.7297 - accuracy:
0.0399 - val_loss: 4.1328 - val_accuracy: 0.0268
```

Epoch 2/400

```
31/31 [=====] - 34s 1s/step - loss: 3.5383 - accuracy:
0.1537 - val_loss: 4.2610 - val_accuracy: 0.0179
```

Epoch 3/400

31/31 [=====] - 35s 1s/step - loss: 2.9516 - accuracy:  
0.2491 - val\_loss: 4.5346 - val\_accuracy: 0.0179  
Epoch 4/400  
31/31 [=====] - 35s 1s/step - loss: 2.4749 - accuracy:  
0.3632 - val\_loss: 4.8323 - val\_accuracy: 0.0179  
Epoch 5/400  
31/31 [=====] - 34s 1s/step - loss: 2.1175 - accuracy:  
0.4701 - val\_loss: 5.0776 - val\_accuracy: 0.0179  
Epoch 6/400  
31/31 [=====] - 34s 1s/step - loss: 1.8282 - accuracy:  
0.5311 - val\_loss: 5.4448 - val\_accuracy: 0.0179  
Epoch 7/400  
31/31 [=====] - 34s 1s/step - loss: 1.6815 - accuracy:  
0.5558 - val\_loss: 5.6547 - val\_accuracy: 0.0156  
Epoch 8/400  
31/31 [=====] - 34s 1s/step - loss: 1.3730 - accuracy:  
0.6490 - val\_loss: 5.9653 - val\_accuracy: 0.0201  
Epoch 9/400  
31/31 [=====] - 34s 1s/step - loss: 1.2582 - accuracy:  
0.6807 - val\_loss: 6.1347 - val\_accuracy: 0.0201  
Epoch 10/400  
31/31 [=====] - 34s 1s/step - loss: 1.0790 - accuracy:  
0.7250 - val\_loss: 6.0689 - val\_accuracy: 0.0201  
Epoch 11/400  
31/31 [=====] - 34s 1s/step - loss: 0.9504 - accuracy:  
0.7733 - val\_loss: 5.8072 - val\_accuracy: 0.0246  
Epoch 12/400  
31/31 [=====] - 34s 1s/step - loss: 0.8048 - accuracy:  
0.8265 - val\_loss: 5.8509 - val\_accuracy: 0.0201  
Epoch 13/400  
31/31 [=====] - 34s 1s/step - loss: 0.7183 - accuracy:  
0.8360 - val\_loss: 5.5358 - val\_accuracy: 0.0268  
Epoch 14/400  
31/31 [=====] - 34s 1s/step - loss: 0.6578 - accuracy:  
0.8456 - val\_loss: 5.2758 - val\_accuracy: 0.0379  
Epoch 15/400  
31/31 [=====] - 34s 1s/step - loss: 0.5658 - accuracy:  
0.8655 - val\_loss: 4.6730 - val\_accuracy: 0.0714  
Epoch 16/400  
31/31 [=====] - 34s 1s/step - loss: 0.4676 - accuracy:  
0.8971 - val\_loss: 3.8975 - val\_accuracy: 0.1496  
Epoch 17/400  
31/31 [=====] - 34s 1s/step - loss: 0.4240 - accuracy:  
0.9136 - val\_loss: 3.4903 - val\_accuracy: 0.2232  
Epoch 18/400  
31/31 [=====] - 34s 1s/step - loss: 0.3808 - accuracy:  
0.9151 - val\_loss: 3.5444 - val\_accuracy: 0.2589  
Epoch 19/400



31/31 [=====] - 35s 1s/step - loss: 0.3431 - accuracy:  
0.9351 - val\_loss: 2.6495 - val\_accuracy: 0.3683  
Epoch 20/400  
31/31 [=====] - 34s 1s/step - loss: 0.3319 - accuracy:  
0.9314 - val\_loss: 2.2859 - val\_accuracy: 0.4152  
Epoch 21/400  
31/31 [=====] - 34s 1s/step - loss: 0.3103 - accuracy:  
0.9346 - val\_loss: 2.2489 - val\_accuracy: 0.4643  
Epoch 22/400  
31/31 [=====] - 34s 1s/step - loss: 0.2913 - accuracy:  
0.9393 - val\_loss: 2.0367 - val\_accuracy: 0.4978  
Epoch 23/400  
31/31 [=====] - 34s 1s/step - loss: 0.2471 - accuracy:  
0.9489 - val\_loss: 1.9282 - val\_accuracy: 0.5223  
Epoch 24/400  
31/31 [=====] - 34s 1s/step - loss: 0.2128 - accuracy:  
0.9565 - val\_loss: 1.7473 - val\_accuracy: 0.5692  
Epoch 25/400  
31/31 [=====] - 34s 1s/step - loss: 0.2219 - accuracy:  
0.9592 - val\_loss: 2.0040 - val\_accuracy: 0.5335  
Epoch 26/400  
31/31 [=====] - 34s 1s/step - loss: 0.1746 - accuracy:  
0.9658 - val\_loss: 1.7587 - val\_accuracy: 0.5424  
Epoch 27/400  
31/31 [=====] - 34s 1s/step - loss: 0.1728 - accuracy:  
0.9705 - val\_loss: 1.6786 - val\_accuracy: 0.5469  
Epoch 28/400  
31/31 [=====] - 34s 1s/step - loss: 0.1582 - accuracy:  
0.9673 - val\_loss: 1.7511 - val\_accuracy: 0.5580  
Epoch 29/400  
31/31 [=====] - 34s 1s/step - loss: 0.1567 - accuracy:  
0.9695 - val\_loss: 1.7618 - val\_accuracy: 0.5781  
Epoch 30/400  
31/31 [=====] - 34s 1s/step - loss: 0.1358 - accuracy:  
0.9740 - val\_loss: 1.7916 - val\_accuracy: 0.5580  
Epoch 31/400  
31/31 [=====] - 35s 1s/step - loss: 0.1129 - accuracy:  
0.9795 - val\_loss: 1.7857 - val\_accuracy: 0.5558  
Epoch 32/400  
31/31 [=====] - 34s 1s/step - loss: 0.1104 - accuracy:  
0.9858 - val\_loss: 1.7867 - val\_accuracy: 0.5647  
Epoch 33/400  
31/31 [=====] - 34s 1s/step - loss: 0.1180 - accuracy:  
0.9760 - val\_loss: 1.7575 - val\_accuracy: 0.5536  
Epoch 34/400  
31/31 [=====] - 34s 1s/step - loss: 0.1020 - accuracy:  
0.9823 - val\_loss: 1.6768 - val\_accuracy: 0.5781  
Epoch 35/400

31/31 [=====] - 34s 1s/step - loss: 0.1037 - accuracy:  
0.9852 - val\_loss: 1.8017 - val\_accuracy: 0.5625  
Epoch 36/400  
31/31 [=====] - 34s 1s/step - loss: 0.0904 - accuracy:  
0.9817 - val\_loss: 1.7548 - val\_accuracy: 0.5670  
Epoch 37/400  
31/31 [=====] - 34s 1s/step - loss: 0.0850 - accuracy:  
0.9863 - val\_loss: 1.7734 - val\_accuracy: 0.5558  
Epoch 38/400  
31/31 [=====] - 34s 1s/step - loss: 0.0803 - accuracy:  
0.9871 - val\_loss: 1.7862 - val\_accuracy: 0.5580  
Epoch 39/400  
31/31 [=====] - 34s 1s/step - loss: 0.0766 - accuracy:  
0.9893 - val\_loss: 1.7373 - val\_accuracy: 0.5804  
Epoch 40/400  
31/31 [=====] - 34s 1s/step - loss: 0.0851 - accuracy:  
0.9844 - val\_loss: 1.8073 - val\_accuracy: 0.5804  
Epoch 41/400  
31/31 [=====] - 34s 1s/step - loss: 0.0808 - accuracy:  
0.9867 - val\_loss: 1.7614 - val\_accuracy: 0.5804  
Epoch 42/400  
31/31 [=====] - 34s 1s/step - loss: 0.0896 - accuracy:  
0.9763 - val\_loss: 1.8213 - val\_accuracy: 0.5670  
Epoch 43/400  
31/31 [=====] - 34s 1s/step - loss: 0.0644 - accuracy:  
0.9921 - val\_loss: 1.7591 - val\_accuracy: 0.5759  
Epoch 44/400  
31/31 [=====] - 34s 1s/step - loss: 0.0647 - accuracy:  
0.9889 - val\_loss: 1.9004 - val\_accuracy: 0.5781  
Epoch 45/400  
31/31 [=====] - 34s 1s/step - loss: 0.0744 - accuracy:  
0.9847 - val\_loss: 1.8357 - val\_accuracy: 0.5603  
Epoch 46/400  
31/31 [=====] - 34s 1s/step - loss: 0.0603 - accuracy:  
0.9899 - val\_loss: 2.3372 - val\_accuracy: 0.5201  
Epoch 47/400  
31/31 [=====] - 34s 1s/step - loss: 0.0596 - accuracy:  
0.9920 - val\_loss: 1.8935 - val\_accuracy: 0.5625  
Epoch 48/400  
31/31 [=====] - 34s 1s/step - loss: 0.0561 - accuracy:  
0.9887 - val\_loss: 1.9065 - val\_accuracy: 0.5603  
Epoch 49/400  
31/31 [=====] - 34s 1s/step - loss: 0.0620 - accuracy:  
0.9896 - val\_loss: 1.8022 - val\_accuracy: 0.5714  
Epoch 50/400  
31/31 [=====] - 34s 1s/step - loss: 0.0662 - accuracy:  
0.9922 - val\_loss: 1.9641 - val\_accuracy: 0.5558  
Epoch 51/400

```

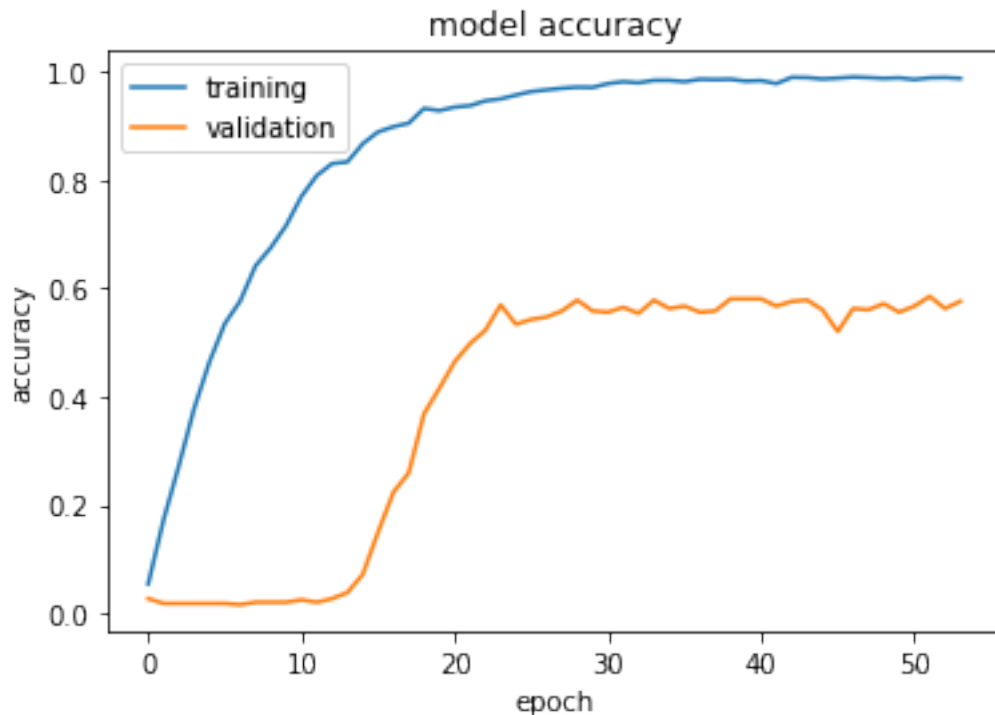
31/31 [=====] - 34s 1s/step - loss: 0.0629 - accuracy:
0.9847 - val_loss: 1.8878 - val_accuracy: 0.5670
Epoch 52/400
31/31 [=====] - 34s 1s/step - loss: 0.0588 - accuracy:
0.9910 - val_loss: 1.8750 - val_accuracy: 0.5848
Epoch 53/400
31/31 [=====] - 34s 1s/step - loss: 0.0488 - accuracy:
0.9928 - val_loss: 1.9083 - val_accuracy: 0.5625
Epoch 54/400
31/31 [=====] - 34s 1s/step - loss: 0.0613 - accuracy:
0.9827 - val_loss: 1.8758 - val_accuracy: 0.5759
Restoring model weights from the end of the best epoch.
Epoch 00054: early stopping

```

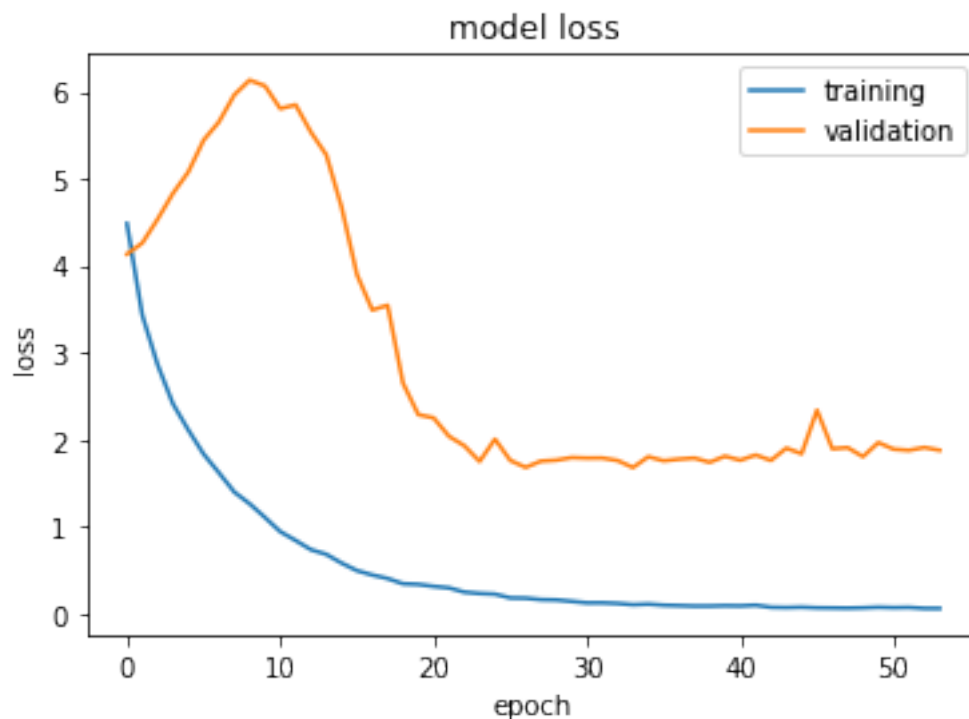
```

[ ]: 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.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()
```



The model performs well on the training set but does not generalize well on the validation set. The model is overfitting on the training data.

One reason for the overfitting can be that there's not enough training data. For this part of the task, this cannot be improved upon. So, I'll try other ways to reduce overfitting: - Data Augmentation - Regularization - Using different activation function - Changing the model architecture

### 1.3.2 Experiment 2: Augmenting training data

I augment the data by randomly shearing it by a range of 0.1 and rotating it by a range of 0.5 degrees.

So, these augmentation methods should help generalize better on unseen images.

**Sidenote: On using other data augmentation techniques** I tried other parameters too, like horizontal and vertical shifts but, they blurred the images and the training set and they didn't really look like the validation samples anymore (I also tested them for a small epoch and they

were actually giving worse results than the first experiment). I'll show some samples to see why I did not augment much on these images, before going forward with the model building.

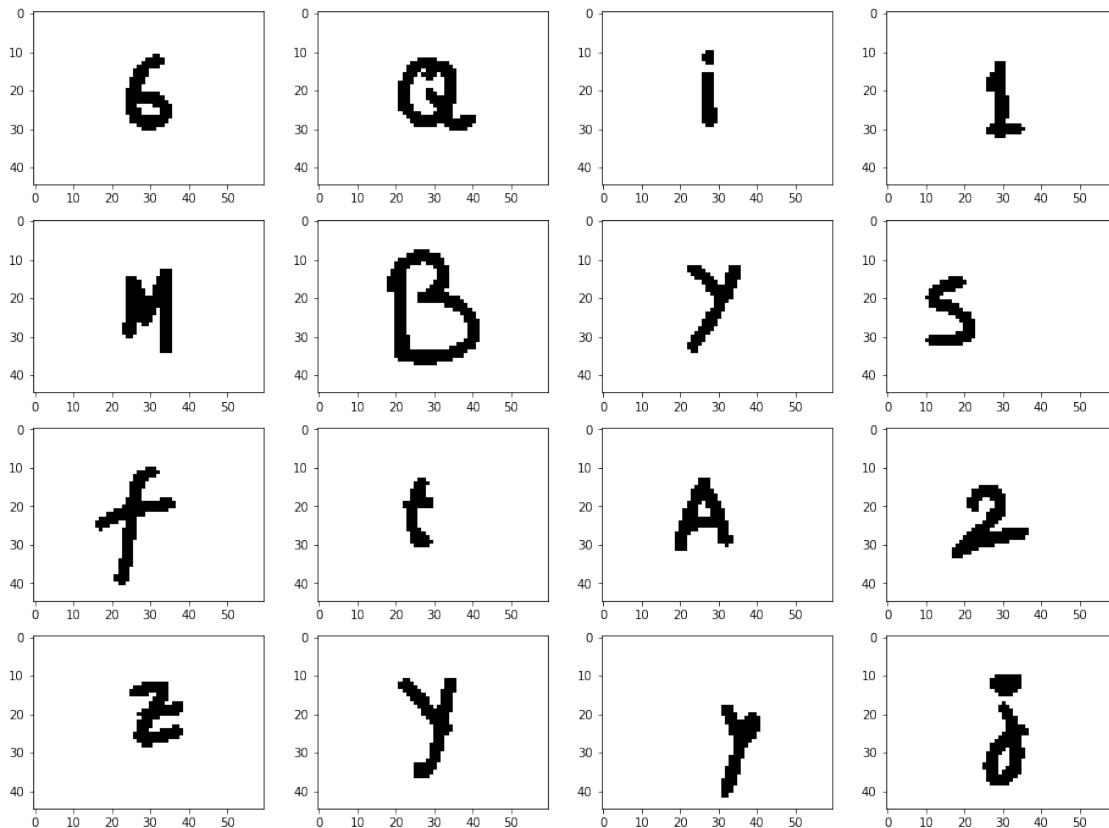
### Unaugmented data

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

no_augmentation_gen = no_augmentation.flow_from_directory(
    '../input/trainpart1zip/train',
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    color_mode='grayscale',
    seed=42,
    shuffle=True)

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

Found 2480 images belonging to 62 classes.



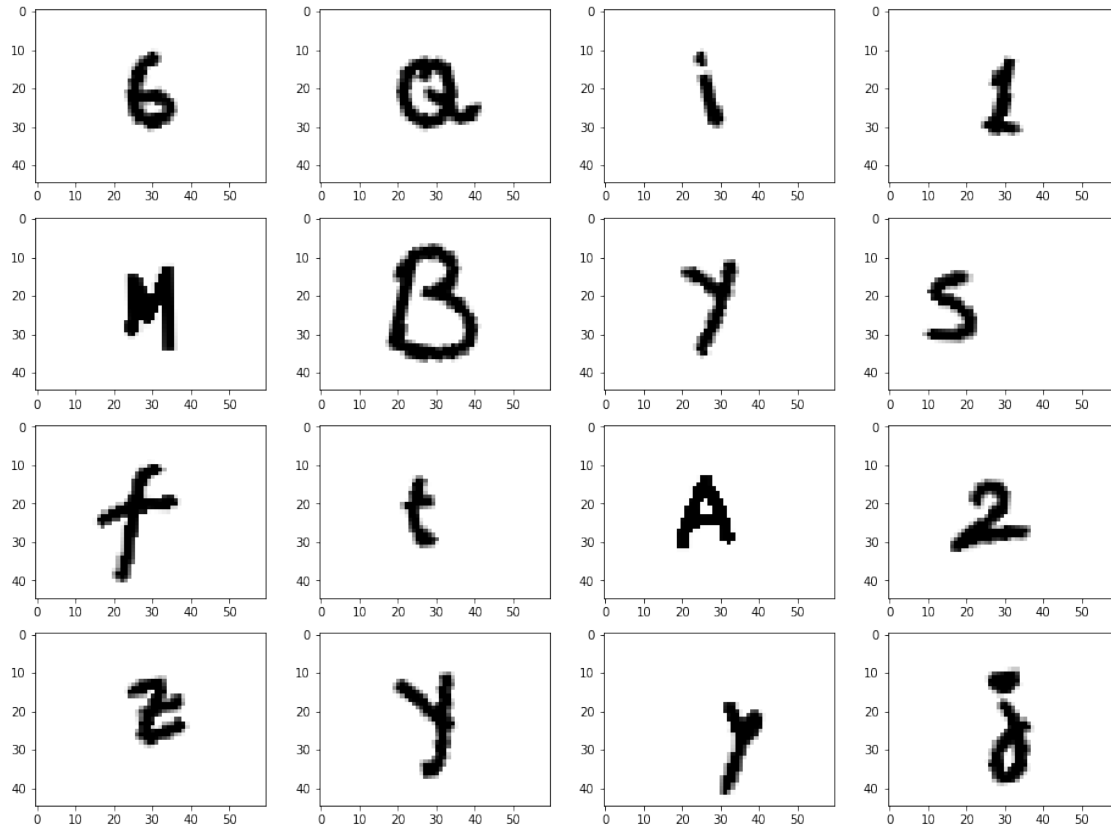
### Checking rotation with max angle of 15

```
[ ]: augmentation_test_rotation = ImageDataGenerator(rescale=1./255,
    ↪rotation_range=15)

augmentation_test_rotation_gen = augmentation_test_rotation.flow_from_directory(
    '../input/trainpart1zip/train',
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    color_mode='grayscale',
    seed=42,
    shuffle=True)

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

Found 2480 images belonging to 62 classes.



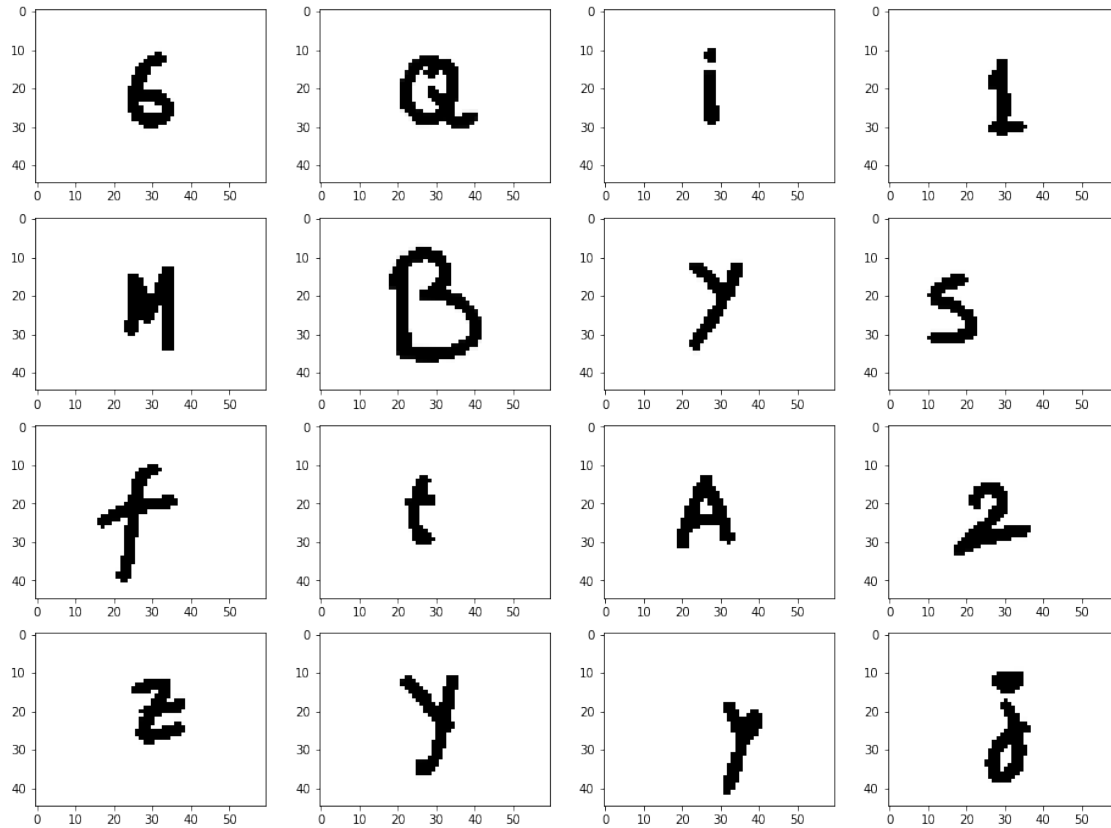
### Checking shear with max shear of 0.3

```
[ ]: augmentation_test_shear = ImageDataGenerator(rescale=1./255, shear_range=0.3)

augmentation_test_shear_gen = augmentation_test_shear.flow_from_directory(
    '../input/trainpart1zip/train',
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    color_mode='grayscale',
    seed=42,
    shuffle=True)

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

Found 2480 images belonging to 62 classes.



### Checking zoom with max zoom of 0.2

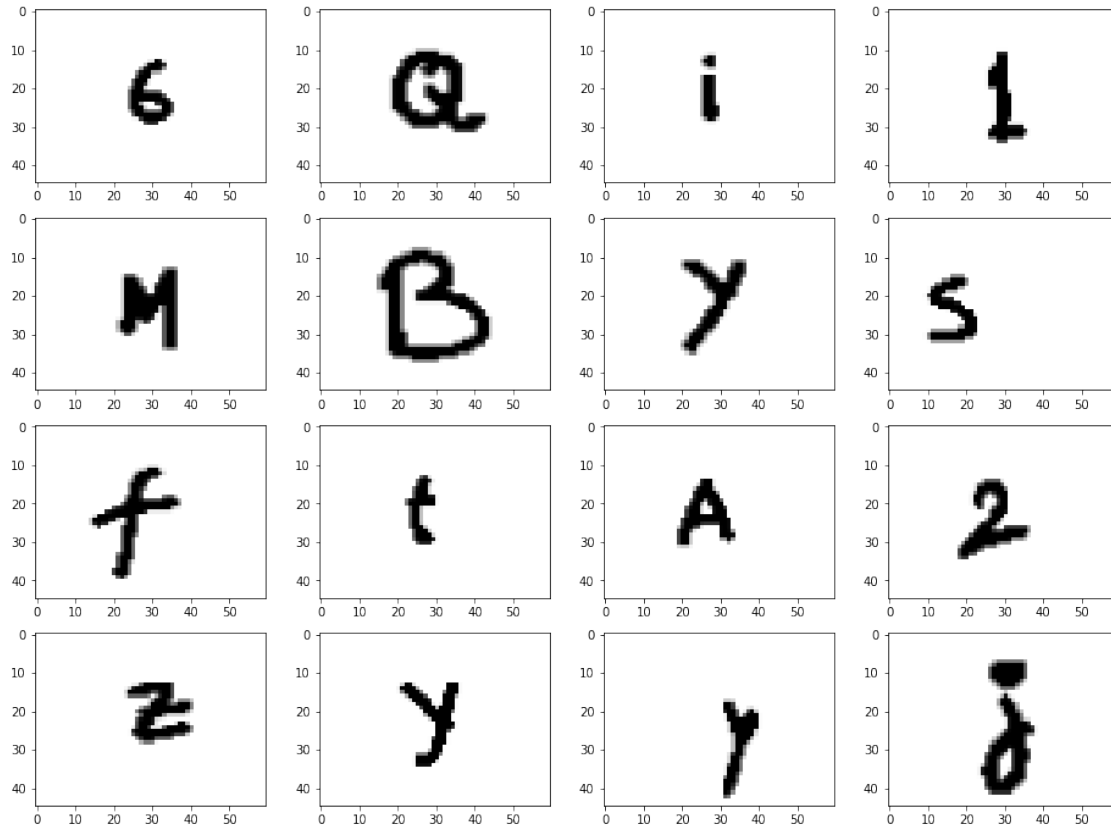
```
[ ]: augmentation_test_zoom = ImageDataGenerator(rescale=1./255, zoom_range=0.2)

augmentation_test_zoom_gen = augmentation_test_zoom.flow_from_directory(
    '../input/trainpart1zip/train',
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    color_mode='grayscale',
    seed=42,
    shuffle=True)

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

Found 2480 images belonging to 62 classes.





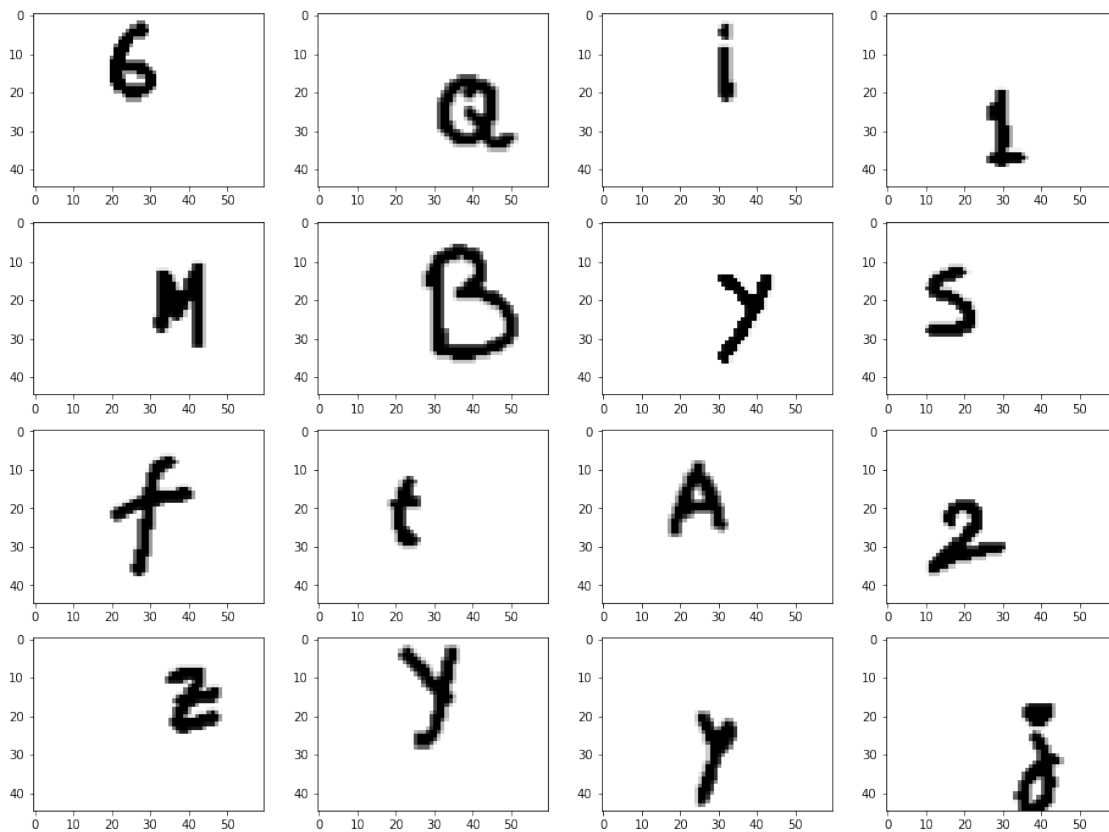
Checking with horizontal and vertical shift of 0.2

```
[ ]: augmentation_test_shift = ImageDataGenerator(rescale=1./255,
    ↳width_shift_range=0.2, height_shift_range=0.2)

augmentation_test_shift_gen = augmentation_test_shift.flow_from_directory(
    '../input/trainpart1zip/train',
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    color_mode='grayscale',
    seed=42,
    shuffle=True)

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

Found 2480 images belonging to 62 classes.



**Conclusion: On what augmentation to choose** These augmentations don't turn out very well. They look very different from the training data. I think the closest is the shear, but there's no noticeable effect that I can see from the naked eye. I think I'll still go on with a little of shear and a little of rotation and see how the model performs.

```
[ ]: train_datagen2 = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    shear_range=0.1,
    rotation_range=0.5)

validation_datagen2 = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2)

[ ]: train_generator2 = train_datagen2.flow_from_directory(
    '../input/trainpart1zip/train',
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
```

```
class_mode='categorical',  
color_mode='grayscale',  
subset='training',  
seed=42,  
shuffle=True)
```

Found 1984 images belonging to 62 classes.

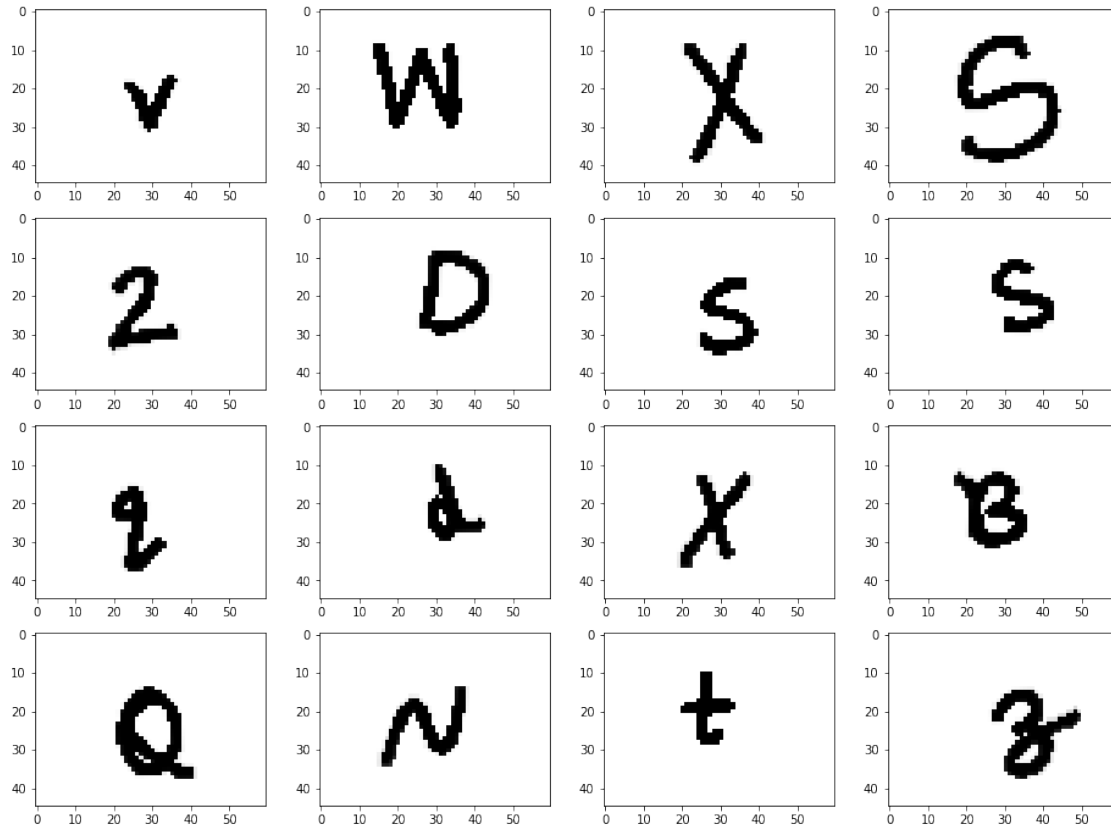
```
[ ]: validation_generator2 = validation_datagen2.flow_from_directory(  
    '../input/trainpart1zip/train',  
    target_size=IMAGE_SIZE,  
    batch_size=BATCH_SIZE,  
    class_mode='categorical',  
    color_mode='grayscale',  
    subset='validation',  
    seed=42,  
    shuffle=True)
```

Found 496 images belonging to 62 classes.

Viewing the Generated samples

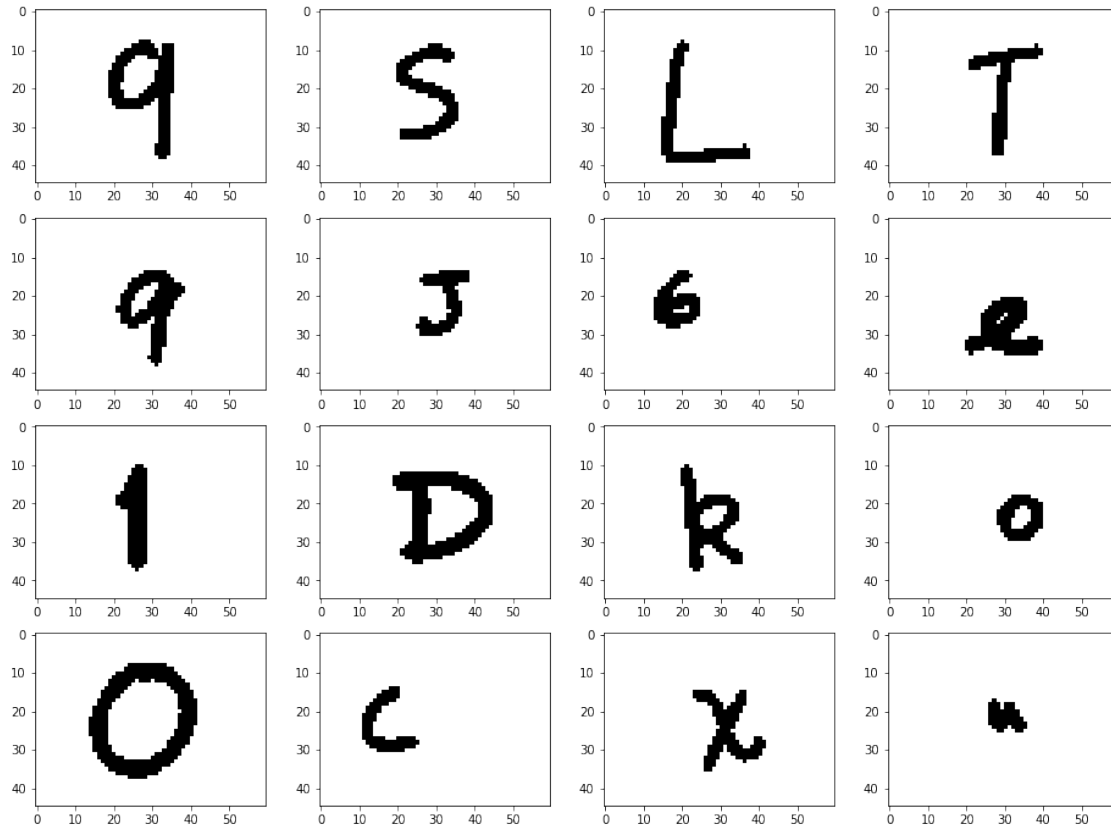
```
[ ]: X_train_batch0, y_train_batch0 = train_generator2.next()  
print(X_train_batch0.shape, y_train_batch0.shape)  
plt.figure(figsize=(16,12))  
for i in range(1, 17):  
    plt.subplot(4,4,i)  
    imshow(tf.squeeze(X_train_batch0[i]), cmap='gray')  
plt.show()
```

(64, 45, 60, 1) (64, 62)



```
[ ]: X_validation_batch0, y_validation_batch0 = validation_generator2.next()
print(X_validation_batch0.shape, y_validation_batch0.shape)
plt.figure(figsize=(16,12))
for i in range(1, 17):
    plt.subplot(4,4,i)
    imshow(tf.squeeze(X_validation_batch0[i]), cmap='gray')
plt.show()
```

(64, 45, 60, 1) (64, 62)



Using the same architecture as before

```
[ ]: model2 = Sequential()

# 1st Convolution Layer
model2.add(Conv2D(6, input_shape=(*IMAGE_SIZE, 1), kernel_size=(5,5),
padding='same', activation='relu'))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2,2), strides=2))

# 2nd Convolution Layer
model2.add(Conv2D(16, kernel_size=(5,5), activation='relu'))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2,2), strides=2))

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

# 1st Fully Connected Layer
model2.add(Dense(256, activation='relu'))
model2.add(BatchNormalization())
```

```

model2.add(Dropout(0.4))

# 2nd Fully Connected Layer
model2.add(Dense(128, activation='relu'))
model2.add(BatchNormalization())
model2.add(Dropout(0.4))

# Output Layer
model2.add(Dense(62, activation='softmax'))

```

```
[ ]: model2.summary()
```

Model: "sequential\_2"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 45, 60, 6)	156
batch_normalization_8 (Batch Normalization)	(None, 45, 60, 6)	24
max_pooling2d_4 (MaxPooling2D)	(None, 22, 30, 6)	0
conv2d_5 (Conv2D)	(None, 18, 26, 16)	2416
batch_normalization_9 (Batch Normalization)	(None, 18, 26, 16)	64
max_pooling2d_5 (MaxPooling2D)	(None, 9, 13, 16)	0
flatten_2 (Flatten)	(None, 1872)	0
dense_6 (Dense)	(None, 256)	479488
batch_normalization_10 (Batch Normalization)	(None, 256)	1024
dropout_4 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 128)	32896
batch_normalization_11 (Batch Normalization)	(None, 128)	512
dropout_5 (Dropout)	(None, 128)	0
dense_8 (Dense)	(None, 62)	7998

Total params: 524,578  
 Trainable params: 523,766  
 Non-trainable params: 812

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

Saving the checkpoint

```
[ ]: checkpoint_filepath2 = 'exp2/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_generator2,  
    epochs=EPOCHS,  
    validation_data=validation_generator2,  
    steps_per_epoch = train_generator2.samples // BATCH_SIZE,  
    validation_steps = validation_generator2.samples // BATCH_SIZE,  
    callbacks=[model_checkpoint_callback2, early_stopping_callback]  
)
```

Epoch 1/400

31/31 [=====] - 37s 1s/step - loss: 4.8922 - accuracy:  
0.0325 - val\_loss: 4.1346 - val\_accuracy: 0.0201

Epoch 2/400

31/31 [=====] - 35s 1s/step - loss: 3.6093 - accuracy:  
0.1459 - val\_loss: 4.1228 - val\_accuracy: 0.0290

Epoch 3/400

31/31 [=====] - 35s 1s/step - loss: 3.0967 - accuracy:  
0.2445 - val\_loss: 4.1612 - val\_accuracy: 0.0179

Epoch 4/400

31/31 [=====] - 35s 1s/step - loss: 2.5994 - accuracy:  
0.3480 - val\_loss: 4.2471 - val\_accuracy: 0.0223

Epoch 5/400

31/31 [=====] - 35s 1s/step - loss: 2.3611 - accuracy:  
0.3881 - val\_loss: 4.4078 - val\_accuracy: 0.0268

Epoch 6/400

31/31 [=====] - 35s 1s/step - loss: 2.0162 - accuracy:  
0.4768 - val\_loss: 4.4157 - val\_accuracy: 0.0312

Epoch 7/400

31/31 [=====] - 35s 1s/step - loss: 1.7933 - accuracy:  
0.5240 - val\_loss: 4.4367 - val\_accuracy: 0.0357

Epoch 8/400

31/31 [=====] - 35s 1s/step - loss: 1.5300 - accuracy:  
0.6031 - val\_loss: 4.3013 - val\_accuracy: 0.0402

Epoch 9/400  
31/31 [=====] - 35s 1s/step - loss: 1.4667 - accuracy: 0.6171 - val\_loss: 4.1002 - val\_accuracy: 0.0625  
Epoch 10/400  
31/31 [=====] - 35s 1s/step - loss: 1.2777 - accuracy: 0.6694 - val\_loss: 3.8498 - val\_accuracy: 0.1161  
Epoch 11/400  
31/31 [=====] - 35s 1s/step - loss: 1.0951 - accuracy: 0.7186 - val\_loss: 3.5382 - val\_accuracy: 0.1317  
Epoch 12/400  
31/31 [=====] - 35s 1s/step - loss: 0.9612 - accuracy: 0.7594 - val\_loss: 3.4244 - val\_accuracy: 0.1786  
Epoch 13/400  
31/31 [=====] - 35s 1s/step - loss: 0.8540 - accuracy: 0.7684 - val\_loss: 2.6652 - val\_accuracy: 0.3147  
Epoch 14/400  
31/31 [=====] - 35s 1s/step - loss: 0.7494 - accuracy: 0.8225 - val\_loss: 2.8891 - val\_accuracy: 0.2634  
Epoch 15/400  
31/31 [=====] - 35s 1s/step - loss: 0.6920 - accuracy: 0.8349 - val\_loss: 2.6686 - val\_accuracy: 0.3504  
Epoch 16/400  
31/31 [=====] - 35s 1s/step - loss: 0.5955 - accuracy: 0.8527 - val\_loss: 2.2025 - val\_accuracy: 0.4420  
Epoch 17/400  
31/31 [=====] - 35s 1s/step - loss: 0.4886 - accuracy: 0.8892 - val\_loss: 2.2851 - val\_accuracy: 0.4420  
Epoch 18/400  
31/31 [=====] - 35s 1s/step - loss: 0.4765 - accuracy: 0.8902 - val\_loss: 2.1791 - val\_accuracy: 0.4397  
Epoch 19/400  
31/31 [=====] - 35s 1s/step - loss: 0.4340 - accuracy: 0.8955 - val\_loss: 2.2505 - val\_accuracy: 0.4353  
Epoch 20/400  
31/31 [=====] - 35s 1s/step - loss: 0.3878 - accuracy: 0.9079 - val\_loss: 1.9392 - val\_accuracy: 0.4866  
Epoch 21/400  
31/31 [=====] - 35s 1s/step - loss: 0.3592 - accuracy: 0.9107 - val\_loss: 2.5304 - val\_accuracy: 0.3951  
Epoch 22/400  
31/31 [=====] - 35s 1s/step - loss: 0.3237 - accuracy: 0.9417 - val\_loss: 1.8786 - val\_accuracy: 0.5067  
Epoch 23/400  
31/31 [=====] - 35s 1s/step - loss: 0.2836 - accuracy: 0.9400 - val\_loss: 2.1518 - val\_accuracy: 0.4487  
Epoch 24/400  
31/31 [=====] - 35s 1s/step - loss: 0.2584 - accuracy: 0.9481 - val\_loss: 1.9832 - val\_accuracy: 0.4643



Epoch 25/400  
31/31 [=====] - 35s 1s/step - loss: 0.2542 - accuracy: 0.9443 - val\_loss: 2.1004 - val\_accuracy: 0.4911  
Epoch 26/400  
31/31 [=====] - 35s 1s/step - loss: 0.2368 - accuracy: 0.9513 - val\_loss: 2.8309 - val\_accuracy: 0.3594  
Epoch 27/400  
31/31 [=====] - 35s 1s/step - loss: 0.2171 - accuracy: 0.9552 - val\_loss: 1.9439 - val\_accuracy: 0.4933  
Epoch 28/400  
31/31 [=====] - 35s 1s/step - loss: 0.2030 - accuracy: 0.9621 - val\_loss: 1.8581 - val\_accuracy: 0.5201  
Epoch 29/400  
31/31 [=====] - 35s 1s/step - loss: 0.1912 - accuracy: 0.9569 - val\_loss: 2.3338 - val\_accuracy: 0.4107  
Epoch 30/400  
31/31 [=====] - 35s 1s/step - loss: 0.1776 - accuracy: 0.9610 - val\_loss: 2.3470 - val\_accuracy: 0.4464  
Epoch 31/400  
31/31 [=====] - 35s 1s/step - loss: 0.1876 - accuracy: 0.9665 - val\_loss: 2.1871 - val\_accuracy: 0.4688  
Epoch 32/400  
31/31 [=====] - 35s 1s/step - loss: 0.1733 - accuracy: 0.9631 - val\_loss: 2.0909 - val\_accuracy: 0.5067  
Epoch 33/400  
31/31 [=====] - 35s 1s/step - loss: 0.1565 - accuracy: 0.9613 - val\_loss: 2.3205 - val\_accuracy: 0.4375  
Epoch 34/400  
31/31 [=====] - 36s 1s/step - loss: 0.1285 - accuracy: 0.9763 - val\_loss: 2.0606 - val\_accuracy: 0.4933  
Epoch 35/400  
31/31 [=====] - 35s 1s/step - loss: 0.1391 - accuracy: 0.9703 - val\_loss: 2.0791 - val\_accuracy: 0.4710  
Epoch 36/400  
31/31 [=====] - 35s 1s/step - loss: 0.1356 - accuracy: 0.9708 - val\_loss: 2.1052 - val\_accuracy: 0.4732  
Epoch 37/400  
31/31 [=====] - 35s 1s/step - loss: 0.1227 - accuracy: 0.9763 - val\_loss: 2.7295 - val\_accuracy: 0.4062  
Epoch 38/400  
31/31 [=====] - 35s 1s/step - loss: 0.1429 - accuracy: 0.9722 - val\_loss: 2.3375 - val\_accuracy: 0.4353  
Epoch 39/400  
31/31 [=====] - 35s 1s/step - loss: 0.1105 - accuracy: 0.9727 - val\_loss: 3.5850 - val\_accuracy: 0.3371  
Epoch 40/400  
31/31 [=====] - 35s 1s/step - loss: 0.1190 - accuracy: 0.9701 - val\_loss: 3.0252 - val\_accuracy: 0.3862

```

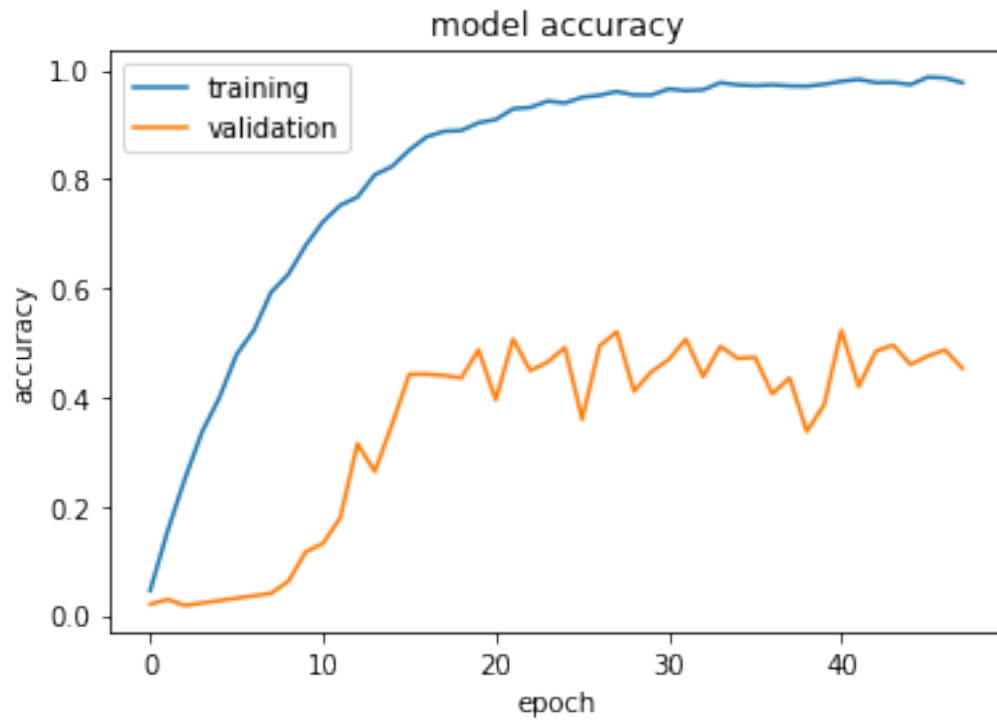
Epoch 41/400
31/31 [=====] - 35s 1s/step - loss: 0.1185 - accuracy:
0.9741 - val_loss: 1.9610 - val_accuracy: 0.5223
Epoch 42/400
31/31 [=====] - 35s 1s/step - loss: 0.0945 - accuracy:
0.9801 - val_loss: 2.6051 - val_accuracy: 0.4196
Epoch 43/400
31/31 [=====] - 35s 1s/step - loss: 0.1067 - accuracy:
0.9750 - val_loss: 2.0708 - val_accuracy: 0.4844
Epoch 44/400
31/31 [=====] - 35s 1s/step - loss: 0.0985 - accuracy:
0.9771 - val_loss: 2.5087 - val_accuracy: 0.4955
Epoch 45/400
31/31 [=====] - 35s 1s/step - loss: 0.0933 - accuracy:
0.9733 - val_loss: 2.4463 - val_accuracy: 0.4598
Epoch 46/400
31/31 [=====] - 35s 1s/step - loss: 0.0832 - accuracy:
0.9854 - val_loss: 2.1240 - val_accuracy: 0.4754
Epoch 47/400
31/31 [=====] - 35s 1s/step - loss: 0.0771 - accuracy:
0.9873 - val_loss: 2.0747 - val_accuracy: 0.4866
Epoch 48/400
31/31 [=====] - 35s 1s/step - loss: 0.0920 - accuracy:
0.9787 - val_loss: 2.4396 - val_accuracy: 0.4531
Restoring model weights from the end of the best epoch.
Epoch 00048: early stopping

```

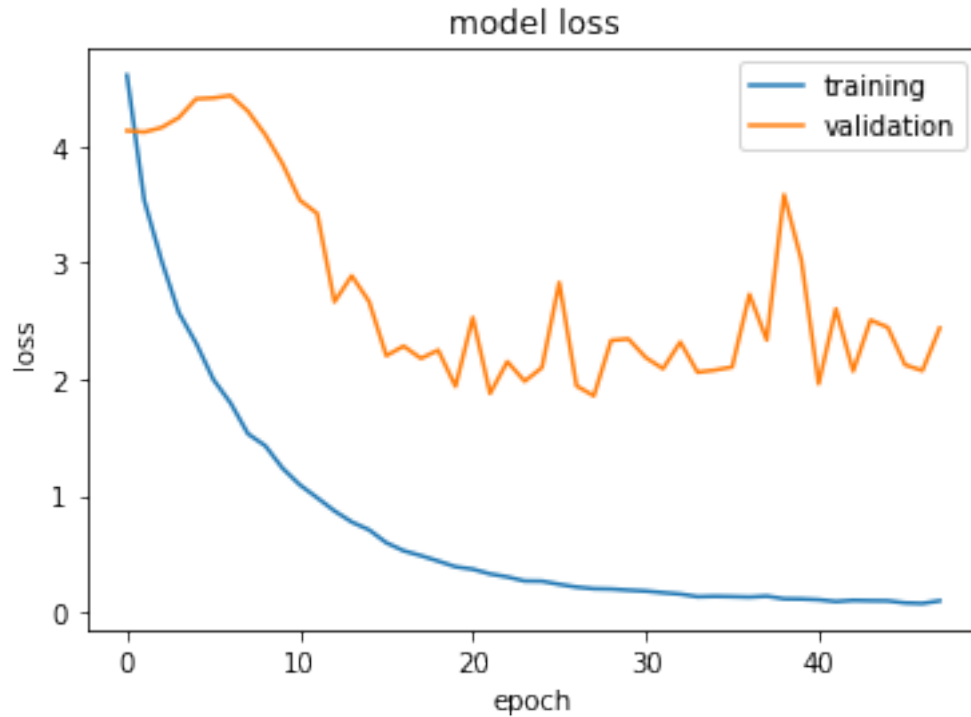
```

[ ]: 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.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()
```



We see that this actually performs worse than the first experiment. I think this is so because in this case, even slightly augmenting data leads to larger variations and since we don't have a lot of training samples, it still overfits to this data and does not generalize well on the validation set.

### 1.3.3 Experiment 3: Using different activation function

**Using Mish Activation** From refs. [4] and [5], I will use the new Mish activation over ReLU. I'll use the training samples from first experiment as they gave better results than the second experiment and will use the same activation.

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

```
[ ]: model3 = Sequential()

# 1st Convolution Layer
model3.add(Conv2D(6, input_shape=(*IMAGE_SIZE, 1),
                    kernel_size=(5,5), padding='same', activation=mish))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2,2), strides=2))

# 2nd Convolution Layer
```

```

model3.add(Conv2D(16, kernel_size=(5,5), activation=mish))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2,2), strides=2))

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

# 1st Fully Connected Layer
model3.add(Dense(256, activation=mish))
model3.add(BatchNormalization())
model3.add(Dropout(0.4))

# 2nd Fully Connected Layer
model3.add(Dense(128, activation=mish))
model3.add(BatchNormalization())
model3.add(Dropout(0.4))

# Output Layer
model3.add(Dense(62, activation='softmax'))

```

```
[ ]: model3.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 45, 60, 6)	156
batch_normalization (Batch Normalization)	(None, 45, 60, 6)	24
max_pooling2d (MaxPooling2D)	(None, 22, 30, 6)	0
conv2d_1 (Conv2D)	(None, 18, 26, 16)	2416
batch_normalization_1 (Batch Normalization)	(None, 18, 26, 16)	64
max_pooling2d_1 (MaxPooling2D)	(None, 9, 13, 16)	0
flatten (Flatten)	(None, 1872)	0
dense (Dense)	(None, 256)	479488
batch_normalization_2 (Batch Normalization)	(None, 256)	1024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32896

```

-----
batch_normalization_3 (Batch Normalization) 512
-----
dropout_1 (Dropout) (None, 128) 0
-----
dense_2 (Dense) (None, 62) 7998
=====
Total params: 524,578
Trainable params: 523,766
Non-trainable params: 812
-----

```

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

Saving the checkpoint

```
[ ]: checkpoint_filepath3 = 'exp3/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_generator1,
    epochs=EPOCHS,
    validation_data=validation_generator1,
    steps_per_epoch = train_generator1.samples // BATCH_SIZE,
    validation_steps = validation_generator1.samples // BATCH_SIZE,
    callbacks=[model_checkpoint_callback3, early_stopping_callback]
)
```

Epoch 1/400

```
31/31 [=====] - 61s 984ms/step - loss: 4.8382 -
accuracy: 0.0325 - val_loss: 4.1591 - val_accuracy: 0.0156
```

Epoch 2/400

```
31/31 [=====] - 30s 973ms/step - loss: 3.5108 -
accuracy: 0.1671 - val_loss: 4.4229 - val_accuracy: 0.0201
```

Epoch 3/400

```
31/31 [=====] - 30s 974ms/step - loss: 2.8792 -
accuracy: 0.2770 - val_loss: 4.8254 - val_accuracy: 0.0179
```

Epoch 4/400

```
31/31 [=====] - 30s 979ms/step - loss: 2.4508 -
accuracy: 0.3762 - val_loss: 5.2647 - val_accuracy: 0.0179
```

Epoch 5/400

```
31/31 [=====] - 30s 974ms/step - loss: 2.1049 -
```

accuracy: 0.4616 - val\_loss: 5.6886 - val\_accuracy: 0.0179  
 Epoch 6/400  
 31/31 [=====] - 30s 974ms/step - loss: 1.8642 -  
 accuracy: 0.5457 - val\_loss: 5.7846 - val\_accuracy: 0.0134  
 Epoch 7/400  
 31/31 [=====] - 30s 969ms/step - loss: 1.5534 -  
 accuracy: 0.6157 - val\_loss: 5.8400 - val\_accuracy: 0.0201  
 Epoch 8/400  
 31/31 [=====] - 30s 972ms/step - loss: 1.3556 -  
 accuracy: 0.6605 - val\_loss: 5.8334 - val\_accuracy: 0.0223  
 Epoch 9/400  
 31/31 [=====] - 30s 975ms/step - loss: 1.1340 -  
 accuracy: 0.7309 - val\_loss: 5.7567 - val\_accuracy: 0.0246  
 Epoch 10/400  
 31/31 [=====] - 30s 971ms/step - loss: 1.0382 -  
 accuracy: 0.7475 - val\_loss: 5.3880 - val\_accuracy: 0.0268  
 Epoch 11/400  
 31/31 [=====] - 30s 975ms/step - loss: 0.8675 -  
 accuracy: 0.7990 - val\_loss: 5.2196 - val\_accuracy: 0.0402  
 Epoch 12/400  
 31/31 [=====] - 30s 975ms/step - loss: 0.7569 -  
 accuracy: 0.8240 - val\_loss: 4.4228 - val\_accuracy: 0.0625  
 Epoch 13/400  
 31/31 [=====] - 30s 972ms/step - loss: 0.6856 -  
 accuracy: 0.8373 - val\_loss: 3.9223 - val\_accuracy: 0.1317  
 Epoch 14/400  
 31/31 [=====] - 30s 971ms/step - loss: 0.5931 -  
 accuracy: 0.8610 - val\_loss: 3.3662 - val\_accuracy: 0.2143  
 Epoch 15/400  
 31/31 [=====] - 30s 973ms/step - loss: 0.5410 -  
 accuracy: 0.8735 - val\_loss: 3.1947 - val\_accuracy: 0.2344  
 Epoch 16/400  
 31/31 [=====] - 30s 972ms/step - loss: 0.4776 -  
 accuracy: 0.8918 - val\_loss: 2.5398 - val\_accuracy: 0.3527  
 Epoch 17/400  
 31/31 [=====] - 30s 974ms/step - loss: 0.4035 -  
 accuracy: 0.9192 - val\_loss: 2.2641 - val\_accuracy: 0.4330  
 Epoch 18/400  
 31/31 [=====] - 30s 969ms/step - loss: 0.3627 -  
 accuracy: 0.9253 - val\_loss: 2.1770 - val\_accuracy: 0.4464  
 Epoch 19/400  
 31/31 [=====] - 30s 976ms/step - loss: 0.3120 -  
 accuracy: 0.9305 - val\_loss: 2.0948 - val\_accuracy: 0.4554  
 Epoch 20/400  
 31/31 [=====] - 30s 975ms/step - loss: 0.3011 -  
 accuracy: 0.9381 - val\_loss: 2.0228 - val\_accuracy: 0.4911  
 Epoch 21/400  
 31/31 [=====] - 30s 980ms/step - loss: 0.2495 -

accuracy: 0.9470 - val\_loss: 1.9106 - val\_accuracy: 0.4911  
 Epoch 22/400  
 31/31 [=====] - 30s 969ms/step - loss: 0.2261 -  
 accuracy: 0.9564 - val\_loss: 1.7929 - val\_accuracy: 0.5335  
 Epoch 23/400  
 31/31 [=====] - 30s 973ms/step - loss: 0.2143 -  
 accuracy: 0.9579 - val\_loss: 2.0653 - val\_accuracy: 0.5089  
 Epoch 24/400  
 31/31 [=====] - 30s 978ms/step - loss: 0.2132 -  
 accuracy: 0.9561 - val\_loss: 1.8173 - val\_accuracy: 0.5513  
 Epoch 25/400  
 31/31 [=====] - 30s 975ms/step - loss: 0.1900 -  
 accuracy: 0.9605 - val\_loss: 1.8378 - val\_accuracy: 0.5424  
 Epoch 26/400  
 31/31 [=====] - 30s 970ms/step - loss: 0.1631 -  
 accuracy: 0.9706 - val\_loss: 1.8676 - val\_accuracy: 0.5424  
 Epoch 27/400  
 31/31 [=====] - 30s 972ms/step - loss: 0.1586 -  
 accuracy: 0.9627 - val\_loss: 1.8381 - val\_accuracy: 0.5402  
 Epoch 28/400  
 31/31 [=====] - 30s 977ms/step - loss: 0.1482 -  
 accuracy: 0.9689 - val\_loss: 1.6875 - val\_accuracy: 0.5848  
 Epoch 29/400  
 31/31 [=====] - 30s 978ms/step - loss: 0.1289 -  
 accuracy: 0.9766 - val\_loss: 2.3107 - val\_accuracy: 0.4621  
 Epoch 30/400  
 31/31 [=====] - 30s 977ms/step - loss: 0.1297 -  
 accuracy: 0.9744 - val\_loss: 1.8469 - val\_accuracy: 0.5402  
 Epoch 31/400  
 31/31 [=====] - 30s 982ms/step - loss: 0.1049 -  
 accuracy: 0.9822 - val\_loss: 1.8450 - val\_accuracy: 0.5714  
 Epoch 32/400  
 31/31 [=====] - 30s 973ms/step - loss: 0.1050 -  
 accuracy: 0.9804 - val\_loss: 1.8608 - val\_accuracy: 0.5692  
 Epoch 33/400  
 31/31 [=====] - 30s 975ms/step - loss: 0.0973 -  
 accuracy: 0.9831 - val\_loss: 1.7893 - val\_accuracy: 0.5491  
 Epoch 34/400  
 31/31 [=====] - 30s 978ms/step - loss: 0.0946 -  
 accuracy: 0.9812 - val\_loss: 1.7682 - val\_accuracy: 0.5714  
 Epoch 35/400  
 31/31 [=====] - 30s 973ms/step - loss: 0.0927 -  
 accuracy: 0.9788 - val\_loss: 1.8935 - val\_accuracy: 0.5647  
 Epoch 36/400  
 31/31 [=====] - 30s 976ms/step - loss: 0.0775 -  
 accuracy: 0.9832 - val\_loss: 2.9494 - val\_accuracy: 0.3951  
 Epoch 37/400  
 31/31 [=====] - 30s 969ms/step - loss: 0.0749 -



```

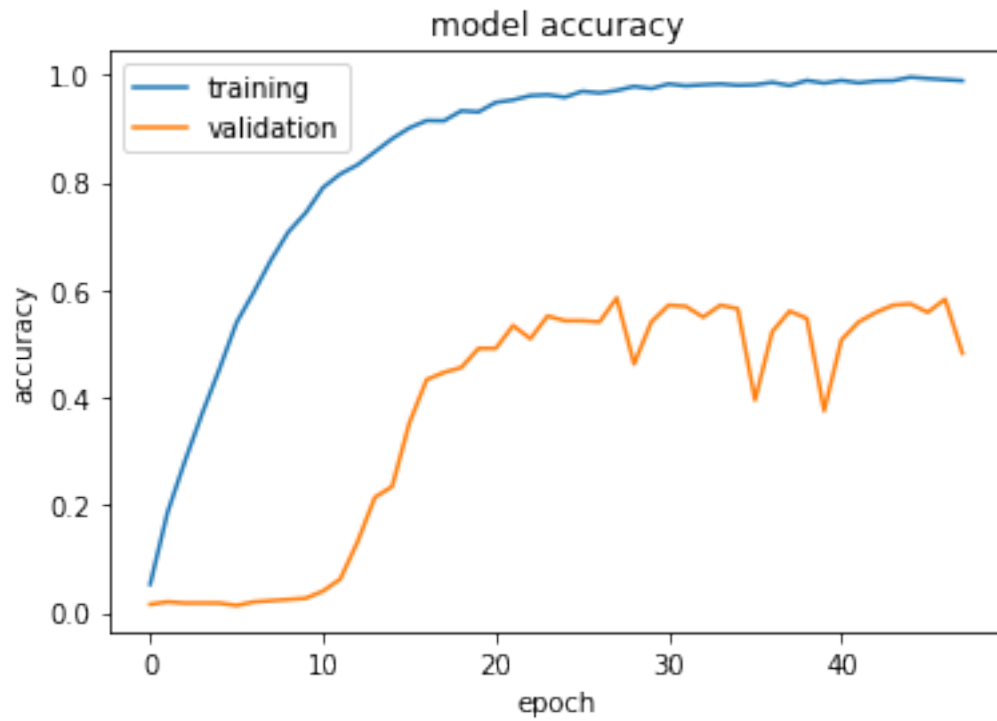
accuracy: 0.9908 - val_loss: 2.1115 - val_accuracy: 0.5223
Epoch 38/400
31/31 [=====] - 30s 971ms/step - loss: 0.0970 -
accuracy: 0.9828 - val_loss: 1.8960 - val_accuracy: 0.5603
Epoch 39/400
31/31 [=====] - 30s 976ms/step - loss: 0.0695 -
accuracy: 0.9889 - val_loss: 1.9347 - val_accuracy: 0.5469
Epoch 40/400
31/31 [=====] - 30s 979ms/step - loss: 0.0806 -
accuracy: 0.9848 - val_loss: 3.0861 - val_accuracy: 0.3750
Epoch 41/400
31/31 [=====] - 30s 972ms/step - loss: 0.0650 -
accuracy: 0.9918 - val_loss: 2.1020 - val_accuracy: 0.5067
Epoch 42/400
31/31 [=====] - 30s 977ms/step - loss: 0.0664 -
accuracy: 0.9845 - val_loss: 1.9804 - val_accuracy: 0.5402
Epoch 43/400
31/31 [=====] - 30s 968ms/step - loss: 0.0686 -
accuracy: 0.9859 - val_loss: 1.8390 - val_accuracy: 0.5580
Epoch 44/400
31/31 [=====] - 30s 978ms/step - loss: 0.0598 -
accuracy: 0.9881 - val_loss: 1.8656 - val_accuracy: 0.5714
Epoch 45/400
31/31 [=====] - 30s 976ms/step - loss: 0.0538 -
accuracy: 0.9946 - val_loss: 1.8879 - val_accuracy: 0.5737
Epoch 46/400
31/31 [=====] - 30s 972ms/step - loss: 0.0532 -
accuracy: 0.9894 - val_loss: 1.8488 - val_accuracy: 0.5580
Epoch 47/400
31/31 [=====] - 30s 975ms/step - loss: 0.0459 -
accuracy: 0.9925 - val_loss: 1.8268 - val_accuracy: 0.5826
Epoch 48/400
31/31 [=====] - 30s 969ms/step - loss: 0.0443 -
accuracy: 0.9918 - val_loss: 2.3868 - val_accuracy: 0.4821
Restoring model weights from the end of the best epoch.
Epoch 00048: early stopping

```

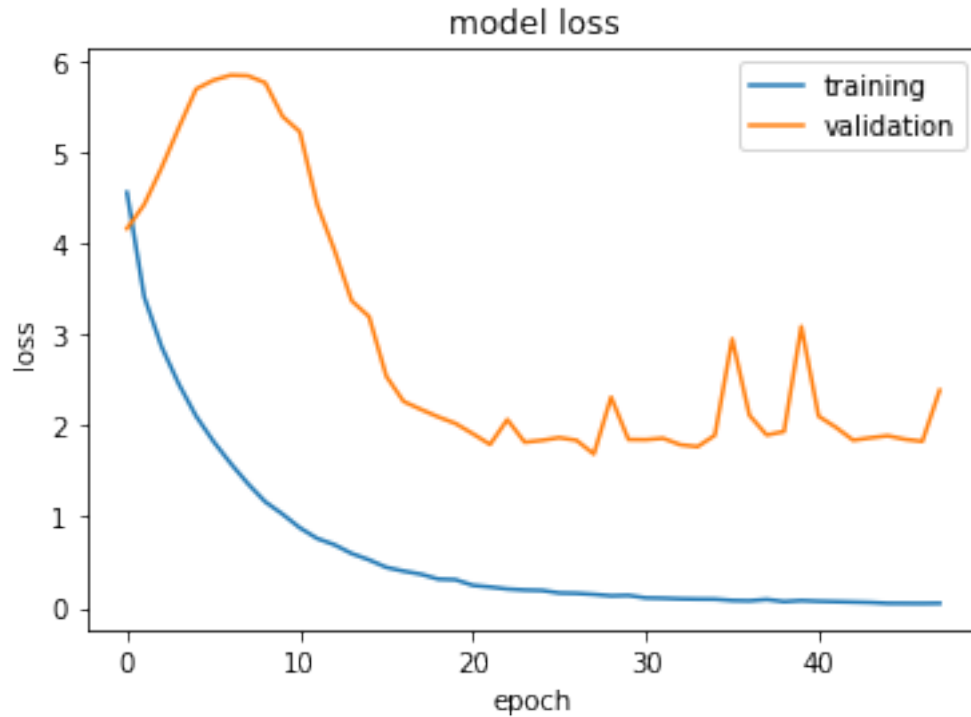
```

[ ]: 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.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()
```



This model performs “slightly” better than our model from Experiment 1 and also does so in less epochs. However, there is some variance as the model overfits. We used early stopping with patience level 20, so our model gives best results at around epoch 28.

**Experiment 7: Using higher temperature in Softmax** Higher temperature in softmax helps get ‘softer’ weights. This is actually the last experiment that I perform here, but put it here in the middle because it fits this section better.

```
[11]: model7 = Sequential()

# 1st Convolution Layer
model7.add(Conv2D(6, input_shape=(*IMAGE_SIZE, 1),
                    kernel_size=(5,5), padding='same', activation=mish))
model7.add(BatchNormalization())
model7.add(MaxPooling2D(pool_size=(2,2), strides=2))

# 2nd Convolution Layer
model7.add(Conv2D(16, kernel_size=(5,5), activation=mish))
model7.add(BatchNormalization())
model7.add(MaxPooling2D(pool_size=(2,2), strides=2))

# Passing to a Fully Connected Layer
model7.add(Flatten())
```

```

# 1st Fully Connected Layer
model7.add(Dense(256, activation=mish))
model7.add(BatchNormalization())
model7.add(Dropout(0.4))

# 2nd Fully Connected Layer
model7.add(Dense(128, activation=mish))
model7.add(BatchNormalization())
model7.add(Dropout(0.4))

# Output Layer
# Increasing the softmax temperature
temp = 5
model7.add(Lambda(lambda x: x / temp))
model7.add(Dense(62, activation='softmax'))

```

```
[12]: model7.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 45, 60, 6)	156
batch_normalization (Batch Normalization)	(None, 45, 60, 6)	24
max_pooling2d (MaxPooling2D)	(None, 22, 30, 6)	0
conv2d_1 (Conv2D)	(None, 18, 26, 16)	2416
batch_normalization_1 (Batch Normalization)	(None, 18, 26, 16)	64
max_pooling2d_1 (MaxPooling2D)	(None, 9, 13, 16)	0
flatten (Flatten)	(None, 1872)	0
dense (Dense)	(None, 256)	479488
batch_normalization_2 (Batch Normalization)	(None, 256)	1024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32896
batch_normalization_3 (Batch Normalization)	(None, 128)	512
dropout_1 (Dropout)	(None, 128)	0

```

-----
lambda (Lambda)                (None, 128)                0
-----
dense_2 (Dense)                 (None, 62)                7998
=====
Total params: 524,578
Trainable params: 523,766
Non-trainable params: 812
-----

```

```
[13]: model7.compile(loss='categorical_crossentropy', optimizer=Adam(),
    ↪ metrics=['accuracy'])
```

```
[18]: checkpoint_filepath7 = 'exp7/checkpoint'
model_checkpoint_callback7 = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath7,
    save_weights_only=True,
    monitor='val_loss',
    mode='min',
    save_best_only=True)
```

```
[19]: history7 = model7.fit(
    train_generator1,
    epochs=EPOCHS,
    validation_data=validation_generator1,
    steps_per_epoch = train_generator1.samples // BATCH_SIZE,
    validation_steps = validation_generator1.samples // BATCH_SIZE,
    callbacks=[model_checkpoint_callback7, early_stopping_callback]
)
```

```

Epoch 1/400
31/31 [=====] - 68s 1s/step - loss: 4.1001 - accuracy:
0.0498 - val_loss: 4.1148 - val_accuracy: 0.0246
Epoch 2/400
31/31 [=====] - 38s 1s/step - loss: 3.7522 - accuracy:
0.1887 - val_loss: 4.1053 - val_accuracy: 0.0179
Epoch 3/400
31/31 [=====] - 38s 1s/step - loss: 3.5069 - accuracy:
0.3088 - val_loss: 4.1128 - val_accuracy: 0.0156
Epoch 4/400
31/31 [=====] - 38s 1s/step - loss: 3.3064 - accuracy:
0.3704 - val_loss: 4.1244 - val_accuracy: 0.0290
Epoch 5/400
31/31 [=====] - 38s 1s/step - loss: 3.0649 - accuracy:
0.4657 - val_loss: 4.1426 - val_accuracy: 0.0290
Epoch 6/400
31/31 [=====] - 38s 1s/step - loss: 2.8262 - accuracy:
0.5386 - val_loss: 4.1336 - val_accuracy: 0.0246

```

Epoch 7/400  
31/31 [=====] - 38s 1s/step - loss: 2.5745 - accuracy: 0.6177 - val\_loss: 4.1471 - val\_accuracy: 0.0290

Epoch 8/400  
31/31 [=====] - 38s 1s/step - loss: 2.3222 - accuracy: 0.6702 - val\_loss: 4.0214 - val\_accuracy: 0.0246

Epoch 9/400  
31/31 [=====] - 38s 1s/step - loss: 2.0725 - accuracy: 0.7416 - val\_loss: 3.9802 - val\_accuracy: 0.0312

Epoch 10/400  
31/31 [=====] - 38s 1s/step - loss: 1.8615 - accuracy: 0.7704 - val\_loss: 3.9252 - val\_accuracy: 0.0491

Epoch 11/400  
31/31 [=====] - 38s 1s/step - loss: 1.6268 - accuracy: 0.8256 - val\_loss: 3.8140 - val\_accuracy: 0.0737

Epoch 12/400  
31/31 [=====] - 38s 1s/step - loss: 1.4081 - accuracy: 0.8648 - val\_loss: 3.7111 - val\_accuracy: 0.1272

Epoch 13/400  
31/31 [=====] - 38s 1s/step - loss: 1.1989 - accuracy: 0.8824 - val\_loss: 3.3292 - val\_accuracy: 0.1830

Epoch 14/400  
31/31 [=====] - 38s 1s/step - loss: 1.0465 - accuracy: 0.9088 - val\_loss: 2.9541 - val\_accuracy: 0.2924

Epoch 15/400  
31/31 [=====] - 38s 1s/step - loss: 0.8774 - accuracy: 0.9250 - val\_loss: 2.9340 - val\_accuracy: 0.2679

Epoch 16/400  
31/31 [=====] - 38s 1s/step - loss: 0.7685 - accuracy: 0.9525 - val\_loss: 2.8449 - val\_accuracy: 0.3036

Epoch 17/400  
31/31 [=====] - 38s 1s/step - loss: 0.6504 - accuracy: 0.9612 - val\_loss: 2.4836 - val\_accuracy: 0.3996

Epoch 18/400  
31/31 [=====] - 38s 1s/step - loss: 0.5525 - accuracy: 0.9636 - val\_loss: 2.3286 - val\_accuracy: 0.4442

Epoch 19/400  
31/31 [=====] - 38s 1s/step - loss: 0.4668 - accuracy: 0.9789 - val\_loss: 1.9924 - val\_accuracy: 0.5000

Epoch 20/400  
31/31 [=====] - 38s 1s/step - loss: 0.4088 - accuracy: 0.9802 - val\_loss: 2.2117 - val\_accuracy: 0.4442

Epoch 21/400  
31/31 [=====] - 38s 1s/step - loss: 0.3612 - accuracy: 0.9881 - val\_loss: 2.1906 - val\_accuracy: 0.4531

Epoch 22/400  
31/31 [=====] - 38s 1s/step - loss: 0.3173 - accuracy: 0.9848 - val\_loss: 1.8808 - val\_accuracy: 0.5179

Epoch 23/400  
31/31 [=====] - 38s 1s/step - loss: 0.2771 - accuracy: 0.9917 - val\_loss: 1.8202 - val\_accuracy: 0.5268  
Epoch 24/400  
31/31 [=====] - 39s 1s/step - loss: 0.2363 - accuracy: 0.9926 - val\_loss: 1.7197 - val\_accuracy: 0.5513  
Epoch 25/400  
31/31 [=====] - 39s 1s/step - loss: 0.2064 - accuracy: 0.9977 - val\_loss: 1.8508 - val\_accuracy: 0.5312  
Epoch 26/400  
31/31 [=====] - 38s 1s/step - loss: 0.1934 - accuracy: 0.9938 - val\_loss: 2.0299 - val\_accuracy: 0.4799  
Epoch 27/400  
31/31 [=====] - 39s 1s/step - loss: 0.1791 - accuracy: 0.9963 - val\_loss: 1.6756 - val\_accuracy: 0.5871  
Epoch 28/400  
31/31 [=====] - 39s 1s/step - loss: 0.1628 - accuracy: 0.9973 - val\_loss: 1.9527 - val\_accuracy: 0.4777  
Epoch 29/400  
31/31 [=====] - 38s 1s/step - loss: 0.1386 - accuracy: 0.9983 - val\_loss: 1.8603 - val\_accuracy: 0.5156  
Epoch 30/400  
31/31 [=====] - 38s 1s/step - loss: 0.1270 - accuracy: 0.9991 - val\_loss: 1.6803 - val\_accuracy: 0.5558  
Epoch 31/400  
31/31 [=====] - 38s 1s/step - loss: 0.1148 - accuracy: 0.9992 - val\_loss: 1.7125 - val\_accuracy: 0.5692  
Epoch 32/400  
31/31 [=====] - 38s 1s/step - loss: 0.0980 - accuracy: 0.9991 - val\_loss: 1.6406 - val\_accuracy: 0.5759  
Epoch 33/400  
31/31 [=====] - 38s 1s/step - loss: 0.0912 - accuracy: 0.9990 - val\_loss: 1.8856 - val\_accuracy: 0.5223  
Epoch 34/400  
31/31 [=====] - 38s 1s/step - loss: 0.0843 - accuracy: 0.9980 - val\_loss: 1.7239 - val\_accuracy: 0.5491  
Epoch 35/400  
31/31 [=====] - 38s 1s/step - loss: 0.0772 - accuracy: 0.9993 - val\_loss: 1.6845 - val\_accuracy: 0.5647  
Epoch 36/400  
31/31 [=====] - 38s 1s/step - loss: 0.0887 - accuracy: 0.9981 - val\_loss: 1.9933 - val\_accuracy: 0.4866  
Epoch 37/400  
31/31 [=====] - 38s 1s/step - loss: 0.0769 - accuracy: 0.9991 - val\_loss: 1.7949 - val\_accuracy: 0.5357  
Epoch 38/400  
31/31 [=====] - 38s 1s/step - loss: 0.0682 - accuracy: 0.9983 - val\_loss: 1.6681 - val\_accuracy: 0.5670

Epoch 39/400  
31/31 [=====] - 37s 1s/step - loss: 0.0610 - accuracy: 0.9990 - val\_loss: 1.8106 - val\_accuracy: 0.5290  
Epoch 40/400  
31/31 [=====] - 37s 1s/step - loss: 0.0524 - accuracy: 0.9999 - val\_loss: 1.7971 - val\_accuracy: 0.5357  
Epoch 41/400  
31/31 [=====] - 37s 1s/step - loss: 0.0572 - accuracy: 0.9965 - val\_loss: 1.7177 - val\_accuracy: 0.5826  
Epoch 42/400  
31/31 [=====] - 37s 1s/step - loss: 0.0493 - accuracy: 1.0000 - val\_loss: 1.9421 - val\_accuracy: 0.4911  
Epoch 43/400  
31/31 [=====] - 37s 1s/step - loss: 0.0437 - accuracy: 1.0000 - val\_loss: 1.7584 - val\_accuracy: 0.5558  
Epoch 44/400  
31/31 [=====] - 37s 1s/step - loss: 0.0471 - accuracy: 0.9995 - val\_loss: 1.6278 - val\_accuracy: 0.5915  
Epoch 45/400  
31/31 [=====] - 37s 1s/step - loss: 0.0418 - accuracy: 0.9991 - val\_loss: 1.8430 - val\_accuracy: 0.5246  
Epoch 46/400  
31/31 [=====] - 37s 1s/step - loss: 0.0430 - accuracy: 0.9993 - val\_loss: 1.7398 - val\_accuracy: 0.5513  
Epoch 47/400  
31/31 [=====] - 36s 1s/step - loss: 0.0394 - accuracy: 1.0000 - val\_loss: 1.6923 - val\_accuracy: 0.5759  
Epoch 48/400  
31/31 [=====] - 36s 1s/step - loss: 0.0364 - accuracy: 0.9994 - val\_loss: 1.7466 - val\_accuracy: 0.5647  
Epoch 49/400  
31/31 [=====] - 36s 1s/step - loss: 0.0362 - accuracy: 1.0000 - val\_loss: 1.6866 - val\_accuracy: 0.5871  
Epoch 50/400  
31/31 [=====] - 36s 1s/step - loss: 0.0342 - accuracy: 0.9999 - val\_loss: 1.7511 - val\_accuracy: 0.5647  
Epoch 51/400  
31/31 [=====] - 36s 1s/step - loss: 0.0340 - accuracy: 0.9994 - val\_loss: 1.6806 - val\_accuracy: 0.5848  
Epoch 52/400  
31/31 [=====] - 36s 1s/step - loss: 0.0329 - accuracy: 0.9994 - val\_loss: 1.6882 - val\_accuracy: 0.5603  
Epoch 53/400  
31/31 [=====] - 36s 1s/step - loss: 0.0292 - accuracy: 0.9983 - val\_loss: 1.7603 - val\_accuracy: 0.5580  
Epoch 54/400  
31/31 [=====] - 36s 1s/step - loss: 0.0316 - accuracy: 0.9994 - val\_loss: 1.8173 - val\_accuracy: 0.5312



```

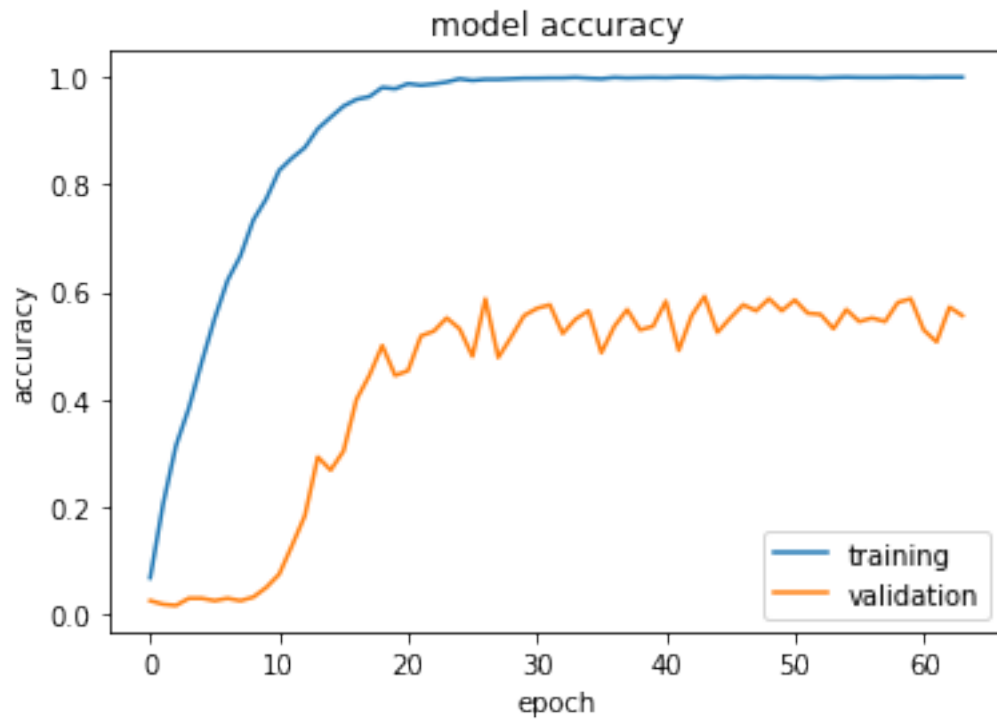
Epoch 55/400
31/31 [=====] - 36s 1s/step - loss: 0.0250 - accuracy:
1.0000 - val_loss: 1.7091 - val_accuracy: 0.5670
Epoch 56/400
31/31 [=====] - 36s 1s/step - loss: 0.0280 - accuracy:
0.9997 - val_loss: 1.7747 - val_accuracy: 0.5446
Epoch 57/400
31/31 [=====] - 36s 1s/step - loss: 0.0244 - accuracy:
0.9999 - val_loss: 1.7784 - val_accuracy: 0.5513
Epoch 58/400
31/31 [=====] - 35s 1s/step - loss: 0.0256 - accuracy:
0.9999 - val_loss: 1.8627 - val_accuracy: 0.5446
Epoch 59/400
31/31 [=====] - 35s 1s/step - loss: 0.0259 - accuracy:
1.0000 - val_loss: 1.6723 - val_accuracy: 0.5804
Epoch 60/400
31/31 [=====] - 36s 1s/step - loss: 0.0241 - accuracy:
1.0000 - val_loss: 1.7835 - val_accuracy: 0.5871
Epoch 61/400
31/31 [=====] - 36s 1s/step - loss: 0.0222 - accuracy:
0.9999 - val_loss: 1.8883 - val_accuracy: 0.5290
Epoch 62/400
31/31 [=====] - 35s 1s/step - loss: 0.0238 - accuracy:
1.0000 - val_loss: 2.1100 - val_accuracy: 0.5067
Epoch 63/400
31/31 [=====] - 35s 1s/step - loss: 0.0196 - accuracy:
1.0000 - val_loss: 1.7393 - val_accuracy: 0.5714
Epoch 64/400
31/31 [=====] - 35s 1s/step - loss: 0.0233 - accuracy:
1.0000 - val_loss: 1.8418 - val_accuracy: 0.5558
Restoring model weights from the end of the best epoch.
Epoch 00064: early stopping

```

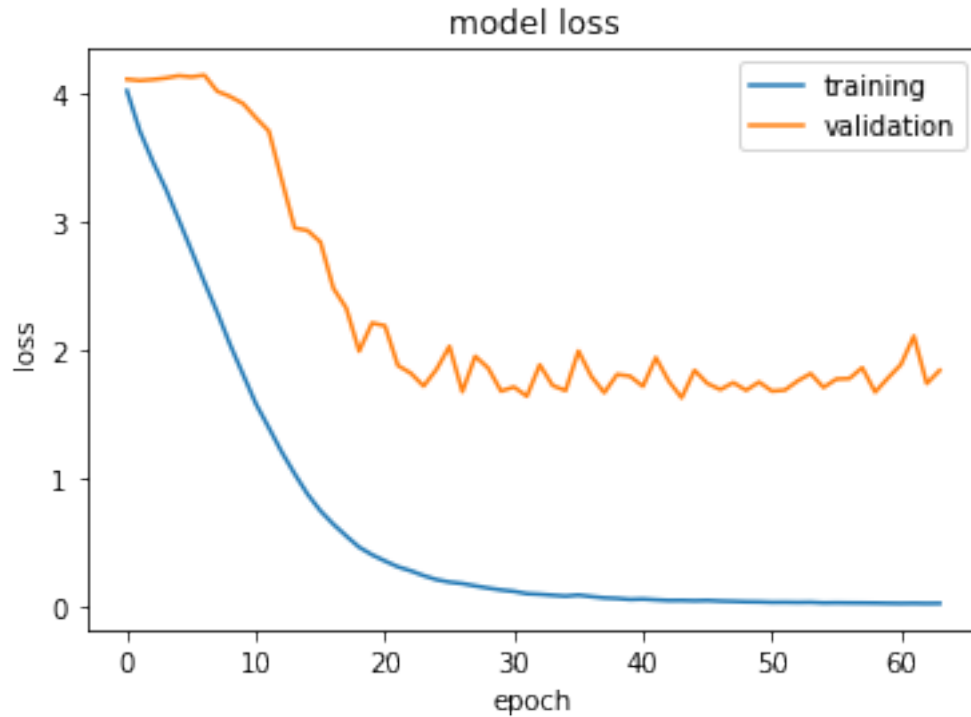
```

[20]: plt.plot(history7.history['accuracy'])
plt.plot(history7.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()

```



```
[21]: plt.plot(history7.history['loss'])
plt.plot(history7.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```



#### 1.3.4 Experiment 4: Changing the Model Architecture

From ref. [1] and [2], I modify the architecture to see if it gives better results. This model actually gives very good results on the MNIST dataset.

```
[ ]: model14 = Sequential()

# 1st Convolution Layer
model14.add(Conv2D(32, input_shape=(*IMAGE_SIZE, 1), kernel_size=3,
    ↪activation=mish))
model14.add(BatchNormalization())
model14.add(Conv2D(32, kernel_size=3, activation=mish))
model14.add(BatchNormalization())
model14.add(Conv2D(32, kernel_size=5, strides=2, padding='same',
    ↪activation=mish))
model14.add(BatchNormalization())
model14.add(Dropout(0.4))

# 2nd Convolution Layer
model14.add(Conv2D(64, kernel_size=3, activation=mish))
model14.add(BatchNormalization())
model14.add(Conv2D(64, kernel_size=3, activation=mish))
model14.add(BatchNormalization())
```

```

model4.add(Conv2D(64, kernel_size=5, strides=2, padding='same',
↳activation=mish))
model4.add(BatchNormalization())
model4.add(Dropout(0.4))

# 3rd Convolution Layer
model4.add(Conv2D(128, kernel_size = 4, activation=mish))
model4.add(BatchNormalization())

# Passing to a Fully Connected Layer
model4.add(Flatten())
model4.add(Dropout(0.4))

# Output Layer
model4.add(Dense(62, activation='softmax'))

```

```
[ ]: model4.summary()
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 43, 58, 32)	320
batch_normalization_4 (Batch Normalization)	(None, 43, 58, 32)	128
conv2d_3 (Conv2D)	(None, 41, 56, 32)	9248
batch_normalization_5 (Batch Normalization)	(None, 41, 56, 32)	128
conv2d_4 (Conv2D)	(None, 21, 28, 32)	25632
batch_normalization_6 (Batch Normalization)	(None, 21, 28, 32)	128
dropout_2 (Dropout)	(None, 21, 28, 32)	0
conv2d_5 (Conv2D)	(None, 19, 26, 64)	18496
batch_normalization_7 (Batch Normalization)	(None, 19, 26, 64)	256
conv2d_6 (Conv2D)	(None, 17, 24, 64)	36928
batch_normalization_8 (Batch Normalization)	(None, 17, 24, 64)	256
conv2d_7 (Conv2D)	(None, 9, 12, 64)	102464
batch_normalization_9 (Batch Normalization)	(None, 9, 12, 64)	256

```

-----
dropout_3 (Dropout)          (None, 9, 12, 64)          0
-----
conv2d_8 (Conv2D)            (None, 6, 9, 128)          131200
-----
batch_normalization_10 (Batc (None, 6, 9, 128)          512
-----
flatten_1 (Flatten)          (None, 6912)                0
-----
dropout_4 (Dropout)          (None, 6912)                0
-----
dense_3 (Dense)              (None, 62)                  428606
=====
Total params: 754,558
Trainable params: 753,726
Non-trainable params: 832
-----

```

```
[ ]: model4.compile(loss='categorical_crossentropy', optimizer=Adam(),
    ↪metrics=['accuracy'])
```

Saving the Checkpoint

```
[ ]: checkpoint_filepath4 = 'exp4/checkpoint'
model_checkpoint_callback4 = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath4,
    save_weights_only=True,
    monitor='val_loss',
    mode='min',
    save_best_only=True)
```

```
[ ]: history4 = model4.fit(
    train_generator1,
    epochs=EPOCHS,
    validation_data=validation_generator1,
    steps_per_epoch = train_generator1.samples // BATCH_SIZE,
    validation_steps = validation_generator1.samples // BATCH_SIZE,
    callbacks=[model_checkpoint_callback4, early_stopping_callback]
)
```

Epoch 1/400

```
31/31 [=====] - 32s 1s/step - loss: 5.3313 - accuracy:
0.0398 - val_loss: 4.6788 - val_accuracy: 0.0156
```

Epoch 2/400

```
31/31 [=====] - 30s 979ms/step - loss: 3.1859 -
accuracy: 0.2657 - val_loss: 5.6807 - val_accuracy: 0.0179
```

Epoch 3/400

```
31/31 [=====] - 30s 979ms/step - loss: 2.0750 -
```

accuracy: 0.4680 - val\_loss: 6.7425 - val\_accuracy: 0.0156  
 Epoch 4/400  
 31/31 [=====] - 30s 974ms/step - loss: 1.4255 -  
 accuracy: 0.6142 - val\_loss: 5.9356 - val\_accuracy: 0.0201  
 Epoch 5/400  
 31/31 [=====] - 30s 974ms/step - loss: 1.0759 -  
 accuracy: 0.6824 - val\_loss: 5.0072 - val\_accuracy: 0.0357  
 Epoch 6/400  
 31/31 [=====] - 30s 983ms/step - loss: 0.7809 -  
 accuracy: 0.7552 - val\_loss: 5.1140 - val\_accuracy: 0.0446  
 Epoch 7/400  
 31/31 [=====] - 30s 973ms/step - loss: 0.5856 -  
 accuracy: 0.8184 - val\_loss: 9.4351 - val\_accuracy: 0.0268  
 Epoch 8/400  
 31/31 [=====] - 30s 975ms/step - loss: 0.4152 -  
 accuracy: 0.8653 - val\_loss: 10.7215 - val\_accuracy: 0.0379  
 Epoch 9/400  
 31/31 [=====] - 30s 973ms/step - loss: 0.3180 -  
 accuracy: 0.8919 - val\_loss: 17.2575 - val\_accuracy: 0.0179  
 Epoch 10/400  
 31/31 [=====] - 30s 976ms/step - loss: 0.2462 -  
 accuracy: 0.9311 - val\_loss: 15.8880 - val\_accuracy: 0.0246  
 Epoch 11/400  
 31/31 [=====] - 30s 977ms/step - loss: 0.2204 -  
 accuracy: 0.9266 - val\_loss: 19.3261 - val\_accuracy: 0.0156  
 Epoch 12/400  
 31/31 [=====] - 30s 976ms/step - loss: 0.2188 -  
 accuracy: 0.9283 - val\_loss: 23.3252 - val\_accuracy: 0.0179  
 Epoch 13/400  
 31/31 [=====] - 30s 982ms/step - loss: 0.1608 -  
 accuracy: 0.9534 - val\_loss: 17.8566 - val\_accuracy: 0.0491  
 Epoch 14/400  
 31/31 [=====] - 30s 977ms/step - loss: 0.1251 -  
 accuracy: 0.9622 - val\_loss: 20.5541 - val\_accuracy: 0.0335  
 Epoch 15/400  
 31/31 [=====] - 30s 973ms/step - loss: 0.1106 -  
 accuracy: 0.9626 - val\_loss: 21.3762 - val\_accuracy: 0.0357  
 Epoch 16/400  
 31/31 [=====] - 30s 986ms/step - loss: 0.1038 -  
 accuracy: 0.9697 - val\_loss: 21.3300 - val\_accuracy: 0.0402  
 Epoch 17/400  
 31/31 [=====] - 30s 977ms/step - loss: 0.0934 -  
 accuracy: 0.9725 - val\_loss: 14.5734 - val\_accuracy: 0.0647  
 Epoch 18/400  
 31/31 [=====] - 30s 977ms/step - loss: 0.1067 -  
 accuracy: 0.9654 - val\_loss: 15.8007 - val\_accuracy: 0.0647  
 Epoch 19/400  
 31/31 [=====] - 30s 982ms/step - loss: 0.0960 -

accuracy: 0.9675 - val\_loss: 4.3696 - val\_accuracy: 0.3527  
 Epoch 20/400  
 31/31 [=====] - 30s 975ms/step - loss: 0.0761 -  
 accuracy: 0.9714 - val\_loss: 4.5411 - val\_accuracy: 0.3281  
 Epoch 21/400  
 31/31 [=====] - 30s 974ms/step - loss: 0.0832 -  
 accuracy: 0.9721 - val\_loss: 5.5686 - val\_accuracy: 0.3013  
 Epoch 22/400  
 31/31 [=====] - 30s 979ms/step - loss: 0.0728 -  
 accuracy: 0.9794 - val\_loss: 2.6852 - val\_accuracy: 0.5156  
 Epoch 23/400  
 31/31 [=====] - 30s 981ms/step - loss: 0.0759 -  
 accuracy: 0.9789 - val\_loss: 2.4136 - val\_accuracy: 0.5402  
 Epoch 24/400  
 31/31 [=====] - 30s 978ms/step - loss: 0.0938 -  
 accuracy: 0.9717 - val\_loss: 2.8570 - val\_accuracy: 0.5290  
 Epoch 25/400  
 31/31 [=====] - 30s 975ms/step - loss: 0.0530 -  
 accuracy: 0.9822 - val\_loss: 3.1094 - val\_accuracy: 0.4911  
 Epoch 26/400  
 31/31 [=====] - 30s 983ms/step - loss: 0.0754 -  
 accuracy: 0.9716 - val\_loss: 6.1126 - val\_accuracy: 0.3058  
 Epoch 27/400  
 31/31 [=====] - 30s 986ms/step - loss: 0.0537 -  
 accuracy: 0.9881 - val\_loss: 6.9506 - val\_accuracy: 0.2879  
 Epoch 28/400  
 31/31 [=====] - 30s 974ms/step - loss: 0.0515 -  
 accuracy: 0.9836 - val\_loss: 3.5170 - val\_accuracy: 0.4799  
 Epoch 29/400  
 31/31 [=====] - 30s 979ms/step - loss: 0.0497 -  
 accuracy: 0.9872 - val\_loss: 2.5949 - val\_accuracy: 0.5580  
 Epoch 30/400  
 31/31 [=====] - 30s 979ms/step - loss: 0.0498 -  
 accuracy: 0.9833 - val\_loss: 3.3020 - val\_accuracy: 0.5000  
 Epoch 31/400  
 31/31 [=====] - 30s 977ms/step - loss: 0.0497 -  
 accuracy: 0.9860 - val\_loss: 3.3340 - val\_accuracy: 0.5112  
 Epoch 32/400  
 31/31 [=====] - 30s 977ms/step - loss: 0.0562 -  
 accuracy: 0.9750 - val\_loss: 4.6205 - val\_accuracy: 0.3906  
 Epoch 33/400  
 31/31 [=====] - 30s 980ms/step - loss: 0.0493 -  
 accuracy: 0.9863 - val\_loss: 6.2758 - val\_accuracy: 0.3259  
 Epoch 34/400  
 31/31 [=====] - 30s 978ms/step - loss: 0.0728 -  
 accuracy: 0.9810 - val\_loss: 3.7161 - val\_accuracy: 0.4286  
 Epoch 35/400  
 31/31 [=====] - 30s 979ms/step - loss: 0.0449 -

```

accuracy: 0.9850 - val_loss: 2.7671 - val_accuracy: 0.5670
Epoch 36/400
31/31 [=====] - 30s 982ms/step - loss: 0.0508 -
accuracy: 0.9801 - val_loss: 5.3039 - val_accuracy: 0.3772
Epoch 37/400
31/31 [=====] - 30s 986ms/step - loss: 0.0696 -
accuracy: 0.9793 - val_loss: 5.5649 - val_accuracy: 0.3504
Epoch 38/400
31/31 [=====] - 30s 977ms/step - loss: 0.0804 -
accuracy: 0.9757 - val_loss: 10.8941 - val_accuracy: 0.2522
Epoch 39/400
31/31 [=====] - 30s 976ms/step - loss: 0.0624 -
accuracy: 0.9783 - val_loss: 3.6520 - val_accuracy: 0.4799
Epoch 40/400
31/31 [=====] - 30s 973ms/step - loss: 0.0436 -
accuracy: 0.9890 - val_loss: 5.3549 - val_accuracy: 0.3906
Epoch 41/400
31/31 [=====] - 30s 976ms/step - loss: 0.0637 -
accuracy: 0.9812 - val_loss: 5.9832 - val_accuracy: 0.3594
Epoch 42/400
31/31 [=====] - 30s 976ms/step - loss: 0.0414 -
accuracy: 0.9890 - val_loss: 5.8161 - val_accuracy: 0.3326
Epoch 43/400
31/31 [=====] - 30s 978ms/step - loss: 0.0526 -
accuracy: 0.9824 - val_loss: 6.0260 - val_accuracy: 0.3415
Restoring model weights from the end of the best epoch.
Epoch 00043: early stopping

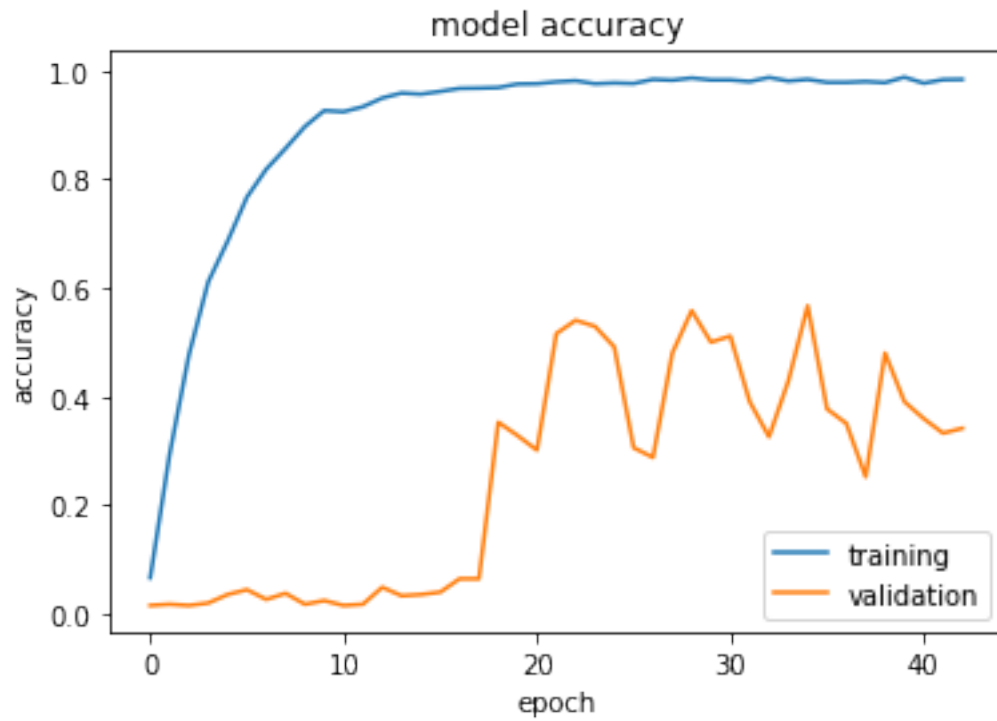
```

```

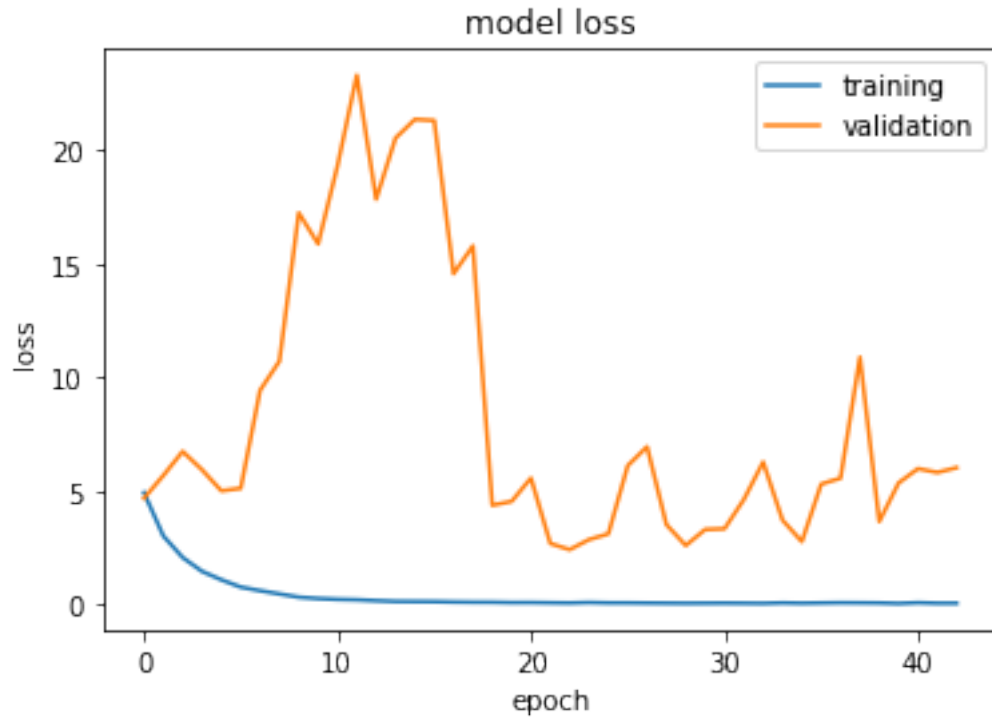
[ ]: plt.plot(history4.history['accuracy'])
plt.plot(history4.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()

```





```
[ ]: plt.plot(history4.history['loss'])
plt.plot(history4.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```



This is actually worse than all other experiments we have done so far. Though the training accuracy converges fast, the validation accuracy and loss have too much variance.

### 1.3.5 Experiment 5: Changing the Model Architecture even further: Using Efficient-net

Efficient Net is being used a lot nowadays so, I'll see how this architecture works for this dataset. I don't have very high hopes for this because: 1. We are not using pre-trained weights 2. We don't have a large trainign data

But I'll still give it a try to see how it does.

For this EfficientNet Model, I would use Early Stopping with larger patience value. This is so because the Keras Website says

“Note: the accuracy will increase very slowly and may overfit.”

on Training a model with EfficientNet from scratch.

Saving the Checkpoint

```
[ ]: checkpoint_filepath5 = 'exp5/checkpoint'
model_checkpoint_callback5 = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath5,
    save_weights_only=True,
    monitor='val_loss',
    mode='min',
```

```
save_best_only=True)
```

Early Stopping Callback

```
[ ]: early_stopping_callback2 = tf.keras.callbacks.EarlyStopping(  
    monitor='val_loss',  
    mode='min',  
    patience=100,  
    restore_best_weights=True,  
    verbose=1)
```

```
[ ]: from tensorflow.keras.applications import EfficientNetB0  
  
# Initializer taken from the source code  
DENSE_KERNEL_INITIALIZER = {  
    'class_name': 'VarianceScaling',  
    'config': {  
        'scale': 1. / 3.,  
        'mode': 'fan_out',  
        'distribution': 'uniform'  
    }  
}  
  
# Input Layer  
inputs = tf.keras.layers.Input(shape=(*IMAGE_SIZE, 1))  
  
# Efficient layer except the top layer  
x = EfficientNetB0(include_top=False, weights=None,  
    input_shape=(*IMAGE_SIZE, 1))(inputs)  
  
# Top  
  
# Global Average Pooling Layer  
x = tf.keras.layers.GlobalAveragePooling2D(name='avg_pool')(x)  
x = tf.keras.layers.Dropout(0.4, name='top_dropout')(x)  
  
# Output Layer  
outputs = tf.keras.layers.Dense(62,  
    activation='softmax',  
    kernel_initializer=DENSE_KERNEL_INITIALIZER,  
    name='predictions')(x)  
  
model5 = tf.keras.Model(inputs, outputs)  
  
model5.compile(  
    optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"]  
)
```

```
model5.summary()
```

Model: "model\_7"

Layer (type)	Output Shape	Param #
input_19 (InputLayer)	[(None, 45, 60, 1)]	0
efficientnetb0 (Functional)	(None, 2, 2, 1280)	4048991
avg_pool (GlobalAveragePooli	(None, 1280)	0
top_dropout (Dropout)	(None, 1280)	0
predictions (Dense)	(None, 62)	79422

Total params: 4,128,413

Trainable params: 4,086,394

Non-trainable params: 42,019

```
[ ]: history5 = model5.fit(
    train_generator1,
    epochs=EPOCHS,
    validation_data=validation_generator1,
    steps_per_epoch = train_generator1.samples // BATCH_SIZE,
    validation_steps = validation_generator1.samples // BATCH_SIZE,
    callbacks=[model_checkpoint_callback5, early_stopping_callback2]
)
```

Epoch 1/400

31/31 [=====] - 42s 1s/step - loss: 4.9151 - accuracy: 0.0201 - val\_loss: 4.1315 - val\_accuracy: 0.0179

Epoch 2/400

31/31 [=====] - 32s 1s/step - loss: 4.4224 - accuracy: 0.0468 - val\_loss: 4.1495 - val\_accuracy: 0.0156

Epoch 3/400

31/31 [=====] - 32s 1s/step - loss: 4.2469 - accuracy: 0.0709 - val\_loss: 4.1952 - val\_accuracy: 0.0179

Epoch 4/400

31/31 [=====] - 32s 1s/step - loss: 3.8510 - accuracy: 0.1198 - val\_loss: 4.2058 - val\_accuracy: 0.0134

Epoch 5/400

31/31 [=====] - 32s 1s/step - loss: 3.4325 - accuracy: 0.2088 - val\_loss: 4.2741 - val\_accuracy: 0.0156

Epoch 6/400

31/31 [=====] - 32s 1s/step - loss: 2.8368 - accuracy:

0.3269 - val\_loss: 4.3717 - val\_accuracy: 0.0179  
Epoch 7/400  
31/31 [=====] - 32s 1s/step - loss: 2.2589 - accuracy:  
0.4527 - val\_loss: 4.7584 - val\_accuracy: 0.0156  
Epoch 8/400  
31/31 [=====] - 32s 1s/step - loss: 1.6094 - accuracy:  
0.5704 - val\_loss: 5.0353 - val\_accuracy: 0.0134  
Epoch 9/400  
31/31 [=====] - 32s 1s/step - loss: 1.2984 - accuracy:  
0.6688 - val\_loss: 5.7188 - val\_accuracy: 0.0179  
Epoch 10/400  
31/31 [=====] - 32s 1s/step - loss: 0.9605 - accuracy:  
0.7401 - val\_loss: 6.1509 - val\_accuracy: 0.0156  
Epoch 11/400  
31/31 [=====] - 32s 1s/step - loss: 0.7479 - accuracy:  
0.7929 - val\_loss: 6.6623 - val\_accuracy: 0.0156  
Epoch 12/400  
31/31 [=====] - 32s 1s/step - loss: 0.6724 - accuracy:  
0.7995 - val\_loss: 7.3675 - val\_accuracy: 0.0156  
Epoch 13/400  
31/31 [=====] - 32s 1s/step - loss: 0.6412 - accuracy:  
0.8292 - val\_loss: 8.9104 - val\_accuracy: 0.0156  
Epoch 14/400  
31/31 [=====] - 32s 1s/step - loss: 0.4667 - accuracy:  
0.8784 - val\_loss: 8.4325 - val\_accuracy: 0.0179  
Epoch 15/400  
31/31 [=====] - 32s 1s/step - loss: 0.4531 - accuracy:  
0.8752 - val\_loss: 8.1772 - val\_accuracy: 0.0156  
Epoch 16/400  
31/31 [=====] - 32s 1s/step - loss: 0.4186 - accuracy:  
0.8964 - val\_loss: 8.1100 - val\_accuracy: 0.0179  
Epoch 17/400  
31/31 [=====] - 32s 1s/step - loss: 0.4693 - accuracy:  
0.8748 - val\_loss: 6.7461 - val\_accuracy: 0.0268  
Epoch 18/400  
31/31 [=====] - 32s 1s/step - loss: 0.4318 - accuracy:  
0.8759 - val\_loss: 6.5141 - val\_accuracy: 0.0179  
Epoch 19/400  
31/31 [=====] - 32s 1s/step - loss: 0.4297 - accuracy:  
0.8865 - val\_loss: 8.9487 - val\_accuracy: 0.0223  
Epoch 20/400  
31/31 [=====] - 32s 1s/step - loss: 0.3713 - accuracy:  
0.8911 - val\_loss: 9.2894 - val\_accuracy: 0.0179  
Epoch 21/400  
31/31 [=====] - 32s 1s/step - loss: 0.4135 - accuracy:  
0.8883 - val\_loss: 9.7851 - val\_accuracy: 0.0179  
Epoch 22/400  
31/31 [=====] - 32s 1s/step - loss: 0.3804 - accuracy:

0.8952 - val\_loss: 8.2647 - val\_accuracy: 0.0246  
 Epoch 23/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2838 - accuracy:  
 0.9314 - val\_loss: 6.5974 - val\_accuracy: 0.0491  
 Epoch 24/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2819 - accuracy:  
 0.9231 - val\_loss: 7.7193 - val\_accuracy: 0.0424  
 Epoch 25/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2607 - accuracy:  
 0.9325 - val\_loss: 6.6000 - val\_accuracy: 0.1183  
 Epoch 26/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2471 - accuracy:  
 0.9353 - val\_loss: 5.7006 - val\_accuracy: 0.1518  
 Epoch 27/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2334 - accuracy:  
 0.9373 - val\_loss: 5.6679 - val\_accuracy: 0.1830  
 Epoch 28/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2431 - accuracy:  
 0.9280 - val\_loss: 5.2927 - val\_accuracy: 0.1920  
 Epoch 29/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2262 - accuracy:  
 0.9401 - val\_loss: 5.1008 - val\_accuracy: 0.2366  
 Epoch 30/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2060 - accuracy:  
 0.9410 - val\_loss: 5.3950 - val\_accuracy: 0.2143  
 Epoch 31/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1982 - accuracy:  
 0.9356 - val\_loss: 6.5190 - val\_accuracy: 0.1585  
 Epoch 32/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2223 - accuracy:  
 0.9415 - val\_loss: 5.8759 - val\_accuracy: 0.2009  
 Epoch 33/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1850 - accuracy:  
 0.9443 - val\_loss: 5.5675 - val\_accuracy: 0.2076  
 Epoch 34/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1874 - accuracy:  
 0.9465 - val\_loss: 5.6272 - val\_accuracy: 0.2188  
 Epoch 35/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2202 - accuracy:  
 0.9338 - val\_loss: 5.9353 - val\_accuracy: 0.1987  
 Epoch 36/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2147 - accuracy:  
 0.9442 - val\_loss: 6.0334 - val\_accuracy: 0.1830  
 Epoch 37/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2092 - accuracy:  
 0.9382 - val\_loss: 6.3359 - val\_accuracy: 0.2188  
 Epoch 38/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2211 - accuracy:

0.9304 - val\_loss: 5.9970 - val\_accuracy: 0.2098  
 Epoch 39/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1915 - accuracy:  
 0.9412 - val\_loss: 6.3312 - val\_accuracy: 0.2143  
 Epoch 40/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2200 - accuracy:  
 0.9326 - val\_loss: 6.4238 - val\_accuracy: 0.2143  
 Epoch 41/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2278 - accuracy:  
 0.9385 - val\_loss: 6.2502 - val\_accuracy: 0.2031  
 Epoch 42/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2487 - accuracy:  
 0.9292 - val\_loss: 5.7950 - val\_accuracy: 0.2522  
 Epoch 43/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2104 - accuracy:  
 0.9415 - val\_loss: 5.5785 - val\_accuracy: 0.2545  
 Epoch 44/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1827 - accuracy:  
 0.9417 - val\_loss: 6.5161 - val\_accuracy: 0.2455  
 Epoch 45/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1858 - accuracy:  
 0.9436 - val\_loss: 6.1442 - val\_accuracy: 0.2031  
 Epoch 46/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1728 - accuracy:  
 0.9513 - val\_loss: 6.6339 - val\_accuracy: 0.1786  
 Epoch 47/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1473 - accuracy:  
 0.9556 - val\_loss: 5.2963 - val\_accuracy: 0.2723  
 Epoch 48/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1677 - accuracy:  
 0.9508 - val\_loss: 6.0020 - val\_accuracy: 0.2433  
 Epoch 49/400  
 31/31 [=====] - 32s 1s/step - loss: 0.2041 - accuracy:  
 0.9449 - val\_loss: 5.8678 - val\_accuracy: 0.2500  
 Epoch 50/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1601 - accuracy:  
 0.9616 - val\_loss: 5.5817 - val\_accuracy: 0.2522  
 Epoch 51/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1253 - accuracy:  
 0.9651 - val\_loss: 5.5550 - val\_accuracy: 0.2656  
 Epoch 52/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1480 - accuracy:  
 0.9604 - val\_loss: 5.1319 - val\_accuracy: 0.2589  
 Epoch 53/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1346 - accuracy:  
 0.9591 - val\_loss: 5.7757 - val\_accuracy: 0.2455  
 Epoch 54/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1710 - accuracy:

0.9507 - val\_loss: 5.8126 - val\_accuracy: 0.2612  
 Epoch 55/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1060 - accuracy:  
 0.9709 - val\_loss: 5.4918 - val\_accuracy: 0.2478  
 Epoch 56/400  
 31/31 [=====] - 32s 1s/step - loss: 0.0894 - accuracy:  
 0.9723 - val\_loss: 5.1332 - val\_accuracy: 0.2589  
 Epoch 57/400  
 31/31 [=====] - 32s 1s/step - loss: 0.0752 - accuracy:  
 0.9770 - val\_loss: 5.1077 - val\_accuracy: 0.2679  
 Epoch 58/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1364 - accuracy:  
 0.9625 - val\_loss: 5.5382 - val\_accuracy: 0.2321  
 Epoch 59/400  
 31/31 [=====] - 32s 1s/step - loss: 0.0782 - accuracy:  
 0.9789 - val\_loss: 5.3676 - val\_accuracy: 0.2522  
 Epoch 60/400  
 31/31 [=====] - 32s 1s/step - loss: 0.0827 - accuracy:  
 0.9771 - val\_loss: 5.4748 - val\_accuracy: 0.2835  
 Epoch 61/400  
 31/31 [=====] - 32s 1s/step - loss: 0.0941 - accuracy:  
 0.9738 - val\_loss: 5.1669 - val\_accuracy: 0.2188  
 Epoch 62/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1117 - accuracy:  
 0.9675 - val\_loss: 5.1975 - val\_accuracy: 0.2656  
 Epoch 63/400  
 31/31 [=====] - 31s 1s/step - loss: 0.0793 - accuracy:  
 0.9790 - val\_loss: 5.4748 - val\_accuracy: 0.2768  
 Epoch 64/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1675 - accuracy:  
 0.9548 - val\_loss: 6.8398 - val\_accuracy: 0.1875  
 Epoch 65/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1590 - accuracy:  
 0.9543 - val\_loss: 6.5336 - val\_accuracy: 0.2321  
 Epoch 66/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1743 - accuracy:  
 0.9508 - val\_loss: 6.0277 - val\_accuracy: 0.2522  
 Epoch 67/400  
 31/31 [=====] - 33s 1s/step - loss: 0.1331 - accuracy:  
 0.9639 - val\_loss: 5.5973 - val\_accuracy: 0.2567  
 Epoch 68/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1496 - accuracy:  
 0.9551 - val\_loss: 5.4199 - val\_accuracy: 0.2522  
 Epoch 69/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1073 - accuracy:  
 0.9657 - val\_loss: 5.3094 - val\_accuracy: 0.2656  
 Epoch 70/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1745 - accuracy:



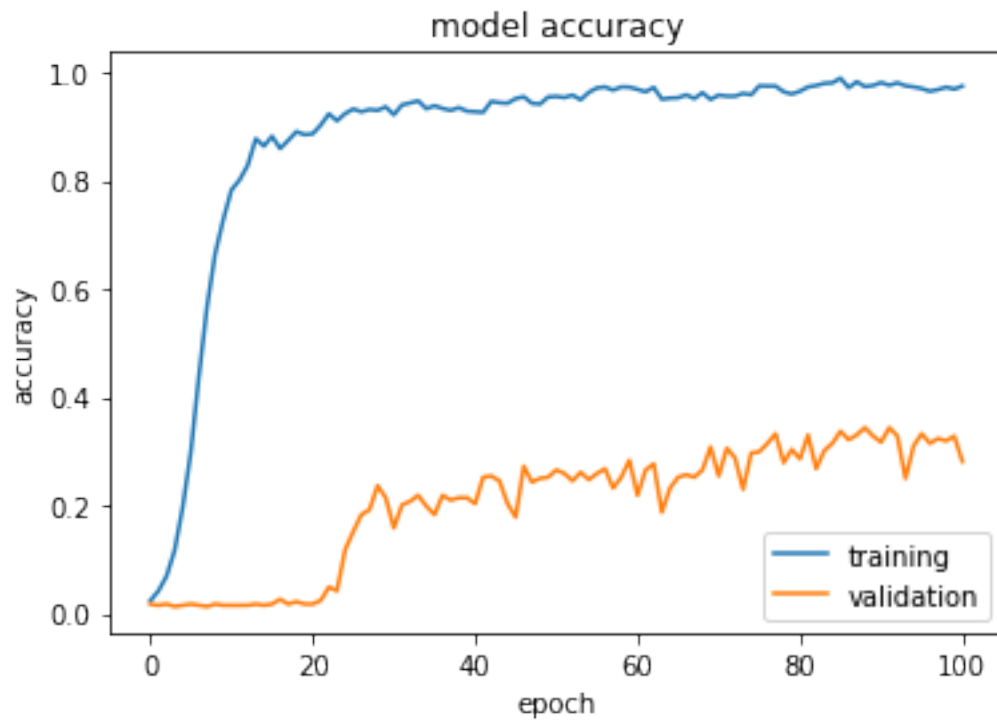
0.9510 - val\_loss: 5.9611 - val\_accuracy: 0.3080  
 Epoch 71/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1436 - accuracy:  
 0.9619 - val\_loss: 6.7030 - val\_accuracy: 0.2545  
 Epoch 72/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1382 - accuracy:  
 0.9589 - val\_loss: 5.7207 - val\_accuracy: 0.3058  
 Epoch 73/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1525 - accuracy:  
 0.9569 - val\_loss: 5.5011 - val\_accuracy: 0.2879  
 Epoch 74/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1293 - accuracy:  
 0.9704 - val\_loss: 6.1242 - val\_accuracy: 0.2299  
 Epoch 75/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1456 - accuracy:  
 0.9605 - val\_loss: 5.5948 - val\_accuracy: 0.2969  
 Epoch 76/400  
 31/31 [=====] - 33s 1s/step - loss: 0.0789 - accuracy:  
 0.9792 - val\_loss: 5.0845 - val\_accuracy: 0.2991  
 Epoch 77/400  
 31/31 [=====] - 33s 1s/step - loss: 0.1044 - accuracy:  
 0.9736 - val\_loss: 5.4756 - val\_accuracy: 0.3147  
 Epoch 78/400  
 31/31 [=====] - 32s 1s/step - loss: 0.0819 - accuracy:  
 0.9782 - val\_loss: 5.4632 - val\_accuracy: 0.3326  
 Epoch 79/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1429 - accuracy:  
 0.9582 - val\_loss: 5.0367 - val\_accuracy: 0.2790  
 Epoch 80/400  
 31/31 [=====] - 32s 1s/step - loss: 0.1364 - accuracy:  
 0.9656 - val\_loss: 5.1341 - val\_accuracy: 0.3036  
 Epoch 81/400  
 31/31 [=====] - 32s 1s/step - loss: 0.0854 - accuracy:  
 0.9695 - val\_loss: 5.0010 - val\_accuracy: 0.2857  
 Epoch 82/400  
 31/31 [=====] - 32s 1s/step - loss: 0.0764 - accuracy:  
 0.9776 - val\_loss: 4.7405 - val\_accuracy: 0.3304  
 Epoch 83/400  
 31/31 [=====] - 32s 1s/step - loss: 0.0855 - accuracy:  
 0.9755 - val\_loss: 5.2778 - val\_accuracy: 0.2679  
 Epoch 84/400  
 31/31 [=====] - 32s 1s/step - loss: 0.0981 - accuracy:  
 0.9753 - val\_loss: 5.1260 - val\_accuracy: 0.3013  
 Epoch 85/400  
 31/31 [=====] - 32s 1s/step - loss: 0.0479 - accuracy:  
 0.9835 - val\_loss: 4.8906 - val\_accuracy: 0.3147  
 Epoch 86/400  
 31/31 [=====] - 31s 1s/step - loss: 0.0572 - accuracy:

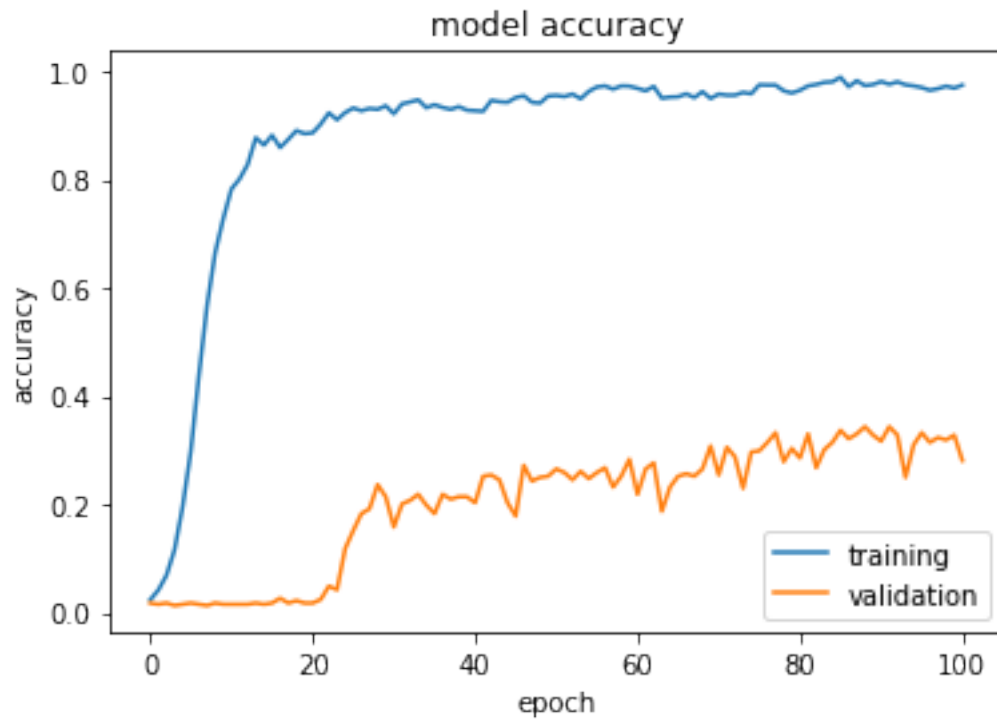
```

0.9886 - val_loss: 4.5963 - val_accuracy: 0.3371
Epoch 87/400
31/31 [=====] - 31s 1s/step - loss: 0.0698 - accuracy:
0.9780 - val_loss: 4.7077 - val_accuracy: 0.3214
Epoch 88/400
31/31 [=====] - 31s 1s/step - loss: 0.0604 - accuracy:
0.9848 - val_loss: 4.8737 - val_accuracy: 0.3304
Epoch 89/400
31/31 [=====] - 31s 1s/step - loss: 0.0778 - accuracy:
0.9751 - val_loss: 4.8525 - val_accuracy: 0.3438
Epoch 90/400
31/31 [=====] - 32s 1s/step - loss: 0.0640 - accuracy:
0.9795 - val_loss: 4.9865 - val_accuracy: 0.3281
Epoch 91/400
31/31 [=====] - 32s 1s/step - loss: 0.0600 - accuracy:
0.9865 - val_loss: 4.9824 - val_accuracy: 0.3170
Epoch 92/400
31/31 [=====] - 32s 1s/step - loss: 0.0794 - accuracy:
0.9811 - val_loss: 5.0791 - val_accuracy: 0.3438
Epoch 93/400
31/31 [=====] - 31s 1s/step - loss: 0.0570 - accuracy:
0.9834 - val_loss: 5.2380 - val_accuracy: 0.3281
Epoch 94/400
31/31 [=====] - 31s 1s/step - loss: 0.1120 - accuracy:
0.9740 - val_loss: 6.1203 - val_accuracy: 0.2500
Epoch 95/400
31/31 [=====] - 32s 1s/step - loss: 0.0696 - accuracy:
0.9751 - val_loss: 5.0995 - val_accuracy: 0.3103
Epoch 96/400
31/31 [=====] - 32s 1s/step - loss: 0.0898 - accuracy:
0.9723 - val_loss: 5.0952 - val_accuracy: 0.3326
Epoch 97/400
31/31 [=====] - 32s 1s/step - loss: 0.1033 - accuracy:
0.9724 - val_loss: 5.6316 - val_accuracy: 0.3147
Epoch 98/400
31/31 [=====] - 32s 1s/step - loss: 0.0979 - accuracy:
0.9698 - val_loss: 5.4321 - val_accuracy: 0.3237
Epoch 99/400
31/31 [=====] - 32s 1s/step - loss: 0.0772 - accuracy:
0.9730 - val_loss: 5.2021 - val_accuracy: 0.3192
Epoch 100/400
31/31 [=====] - 32s 1s/step - loss: 0.0845 - accuracy:
0.9761 - val_loss: 5.2520 - val_accuracy: 0.3281
Epoch 101/400
31/31 [=====] - 32s 1s/step - loss: 0.0814 - accuracy:
0.9800 - val_loss: 5.7342 - val_accuracy: 0.2812
Restoring model weights from the end of the best epoch.
Epoch 00101: early stopping

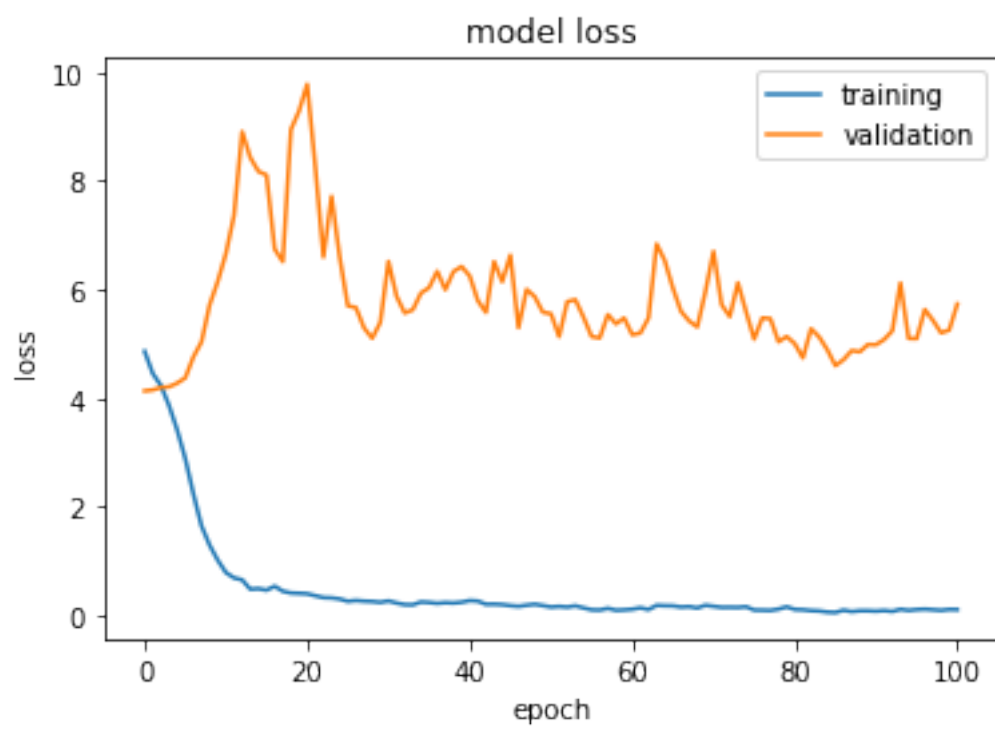
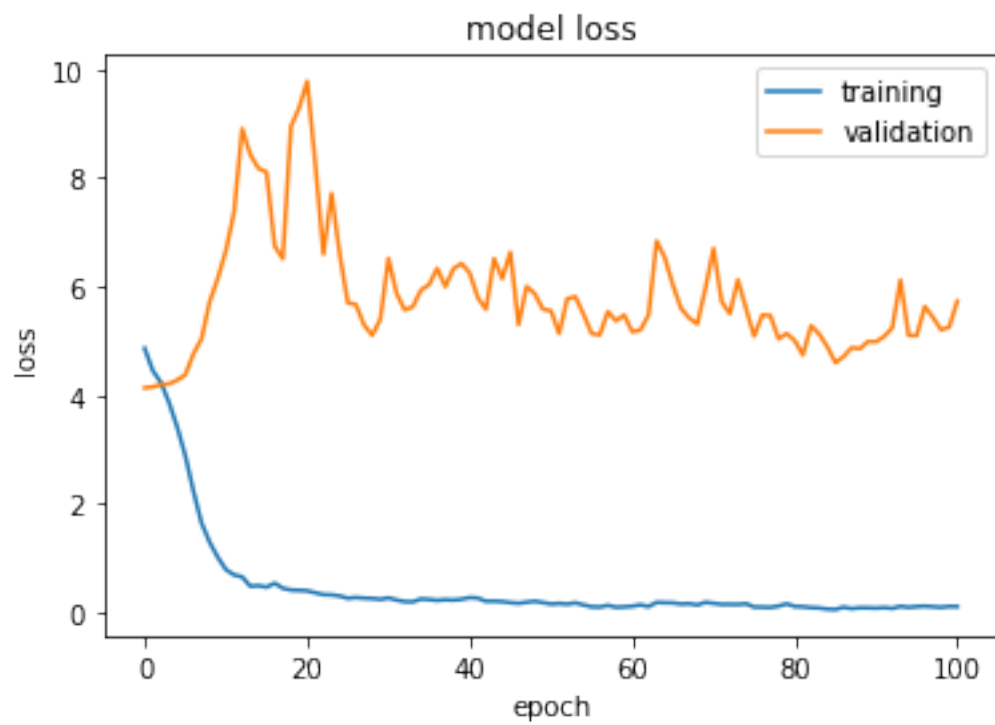
```

```
[ ]: plt.plot(history5.history['accuracy'])
plt.plot(history5.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```





```
[ ]: plt.plot(history5.history['loss'])
plt.plot(history5.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```



As expected, this does not converge. Infact, it's not even reaching a 50% accuracy. This accuracy result is similar to the one shown in ref. [6] [here](#).

### 1.3.6 Experiment 6: Reducing Batch Size

Smaller batch sizes tend to converge faster [12]. Hence, I'll reduce the batch size to 32 to see the performance. I'll use the two best models I have achieved form the past experiments 3 and 4.

#### Architecture from Experiment 3

```
[ ]: BATCH_SIZE = 32
```

```
[ ]: train_generator3 = train_datagen1.flow_from_directory(
    '../input/trainpart1zip/train',
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    color_mode='grayscale',
    subset='training',
    seed=42,
    shuffle=True)

validation_generator3 = train_datagen1.flow_from_directory(
    '../input/trainpart1zip/train',
    target_size=tf.squeeze(IMAGE_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    color_mode='grayscale',
    subset='validation',
    seed=42,
    shuffle=True)
```

Found 1984 images belonging to 62 classes.

Found 496 images belonging to 62 classes.

```
[ ]: model6_1 = Sequential()

# 1st Convolution Layer
model6_1.add(Conv2D(6, input_shape=(*IMAGE_SIZE, 1),
                    kernel_size=(5,5), padding='same', activation=mish))
model6_1.add(BatchNormalization())
model6_1.add(MaxPooling2D(pool_size=(2,2), strides=2))

# 2nd Convolution Layer
model6_1.add(Conv2D(16, kernel_size=(5,5), activation=mish))
model6_1.add(BatchNormalization())
model6_1.add(MaxPooling2D(pool_size=(2,2), strides=2))
```

```

# Passing to a Fully Connected Layer
model6_1.add(Flatten())

# 1st Fully Connected Layer
model6_1.add(Dense(256, activation=mish))
model6_1.add(BatchNormalization())
model6_1.add(Dropout(0.4))

# 2nd Fully Connected Layer
model6_1.add(Dense(128, activation=mish))
model6_1.add(BatchNormalization())
model6_1.add(Dropout(0.4))

# Output Layer
model6_1.add(Dense(62, activation='softmax'))

```

```
[ ]: model6_1.summary()
```

Model: "sequential\_4"

Layer (type)	Output Shape	Param #
conv2d_13 (Conv2D)	(None, 45, 60, 6)	156
batch_normalization_19 (Batch Normalization)	(None, 45, 60, 6)	24
max_pooling2d_6 (MaxPooling2D)	(None, 22, 30, 6)	0
conv2d_14 (Conv2D)	(None, 18, 26, 16)	2416
batch_normalization_20 (Batch Normalization)	(None, 18, 26, 16)	64
max_pooling2d_7 (MaxPooling2D)	(None, 9, 13, 16)	0
flatten_4 (Flatten)	(None, 1872)	0
dense_10 (Dense)	(None, 256)	479488
batch_normalization_21 (Batch Normalization)	(None, 256)	1024
dropout_9 (Dropout)	(None, 256)	0
dense_11 (Dense)	(None, 128)	32896
batch_normalization_22 (Batch Normalization)	(None, 128)	512
dropout_10 (Dropout)	(None, 128)	0

```

-----
dense_12 (Dense)                (None, 62)                7998
=====
Total params: 524,578
Trainable params: 523,766
Non-trainable params: 812
-----

```

```
[ ]: model6_1.compile(loss='categorical_crossentropy', optimizer=Adam(),
    ↪metrics=['accuracy'])
```

Saving the Model Checkpoint

```
[ ]: checkpoint_filepath6_1 = 'exp6_1/checkpoint'
model_checkpoint_callback6_1 = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath6_1,
    save_weights_only=True,
    monitor='val_loss',
    mode='min',
    save_best_only=True)
```

```
[ ]: history6_1 = model6_1.fit(
    train_generator3,
    epochs=EPOCHS,
    validation_data=validation_generator3,
    steps_per_epoch = train_generator3.samples // BATCH_SIZE,
    validation_steps = validation_generator3.samples // BATCH_SIZE,
    callbacks=[model_checkpoint_callback6_1, early_stopping_callback]
)
```

Epoch 1/400

```
62/62 [=====] - 33s 513ms/step - loss: 4.6588 -
accuracy: 0.0335 - val_loss: 4.3612 - val_accuracy: 0.0167
```

Epoch 2/400

```
62/62 [=====] - 31s 499ms/step - loss: 3.3342 -
accuracy: 0.1942 - val_loss: 4.6904 - val_accuracy: 0.0250
```

Epoch 3/400

```
62/62 [=====] - 31s 495ms/step - loss: 2.6940 -
accuracy: 0.3107 - val_loss: 4.8908 - val_accuracy: 0.0208
```

Epoch 4/400

```
62/62 [=====] - 31s 501ms/step - loss: 2.2589 -
accuracy: 0.4113 - val_loss: 4.9160 - val_accuracy: 0.0083
```

Epoch 5/400

```
62/62 [=====] - 30s 493ms/step - loss: 1.8811 -
accuracy: 0.5197 - val_loss: 4.5471 - val_accuracy: 0.0354
```

Epoch 6/400

```
62/62 [=====] - 31s 494ms/step - loss: 1.5784 -
accuracy: 0.5897 - val_loss: 3.7062 - val_accuracy: 0.1250
```



Epoch 7/400  
62/62 [=====] - 31s 496ms/step - loss: 1.3584 - accuracy: 0.6500 - val\_loss: 4.0543 - val\_accuracy: 0.0979

Epoch 8/400  
62/62 [=====] - 30s 492ms/step - loss: 1.1313 - accuracy: 0.7136 - val\_loss: 3.6911 - val\_accuracy: 0.1542

Epoch 9/400  
62/62 [=====] - 31s 494ms/step - loss: 0.9335 - accuracy: 0.7652 - val\_loss: 2.9963 - val\_accuracy: 0.2542

Epoch 10/400  
62/62 [=====] - 31s 502ms/step - loss: 0.8233 - accuracy: 0.7817 - val\_loss: 3.0014 - val\_accuracy: 0.2625

Epoch 11/400  
62/62 [=====] - 31s 498ms/step - loss: 0.6886 - accuracy: 0.8157 - val\_loss: 2.9491 - val\_accuracy: 0.3021

Epoch 12/400  
62/62 [=====] - 31s 503ms/step - loss: 0.5739 - accuracy: 0.8549 - val\_loss: 2.0746 - val\_accuracy: 0.4625

Epoch 13/400  
62/62 [=====] - 31s 498ms/step - loss: 0.5214 - accuracy: 0.8688 - val\_loss: 2.4844 - val\_accuracy: 0.4021

Epoch 14/400  
62/62 [=====] - 31s 504ms/step - loss: 0.4385 - accuracy: 0.8842 - val\_loss: 2.7843 - val\_accuracy: 0.3417

Epoch 15/400  
62/62 [=====] - 31s 503ms/step - loss: 0.3956 - accuracy: 0.9032 - val\_loss: 1.7987 - val\_accuracy: 0.5208

Epoch 16/400  
62/62 [=====] - 31s 503ms/step - loss: 0.3296 - accuracy: 0.9288 - val\_loss: 2.1770 - val\_accuracy: 0.4396

Epoch 17/400  
62/62 [=====] - 31s 508ms/step - loss: 0.3260 - accuracy: 0.9238 - val\_loss: 4.9160 - val\_accuracy: 0.2354

Epoch 18/400  
62/62 [=====] - 31s 502ms/step - loss: 0.2988 - accuracy: 0.9266 - val\_loss: 4.9212 - val\_accuracy: 0.3000

Epoch 19/400  
62/62 [=====] - 31s 501ms/step - loss: 0.2539 - accuracy: 0.9368 - val\_loss: 3.1581 - val\_accuracy: 0.3604

Epoch 20/400  
62/62 [=====] - 31s 501ms/step - loss: 0.2521 - accuracy: 0.9353 - val\_loss: 3.8498 - val\_accuracy: 0.2958

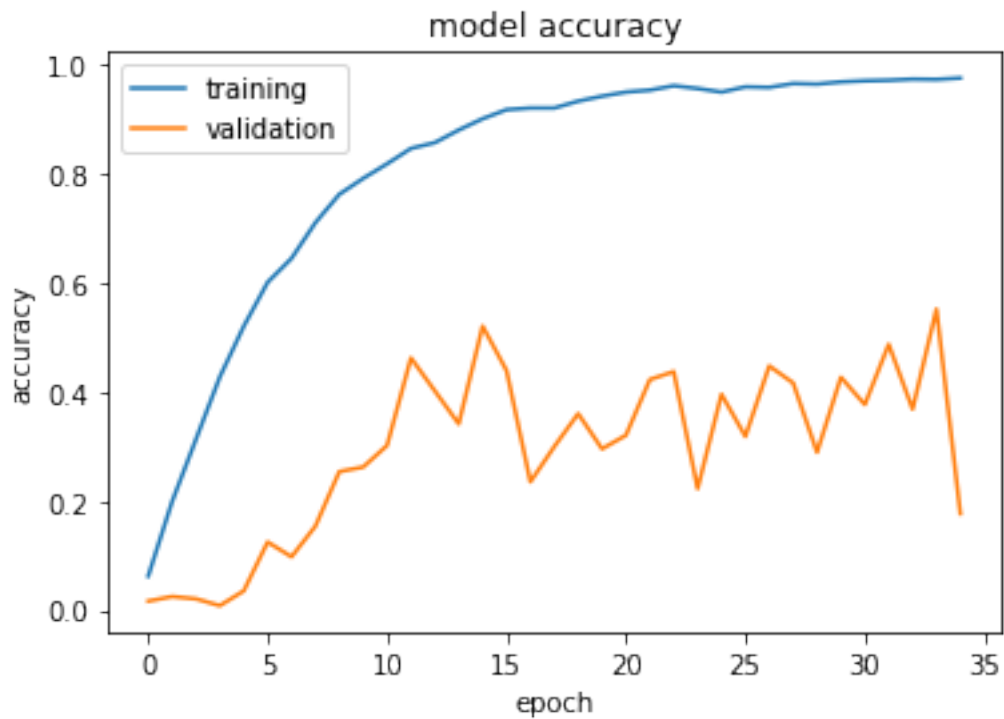
Epoch 21/400  
62/62 [=====] - 31s 504ms/step - loss: 0.2160 - accuracy: 0.9512 - val\_loss: 3.2776 - val\_accuracy: 0.3208

Epoch 22/400  
62/62 [=====] - 31s 507ms/step - loss: 0.2002 - accuracy: 0.9499 - val\_loss: 2.2856 - val\_accuracy: 0.4229

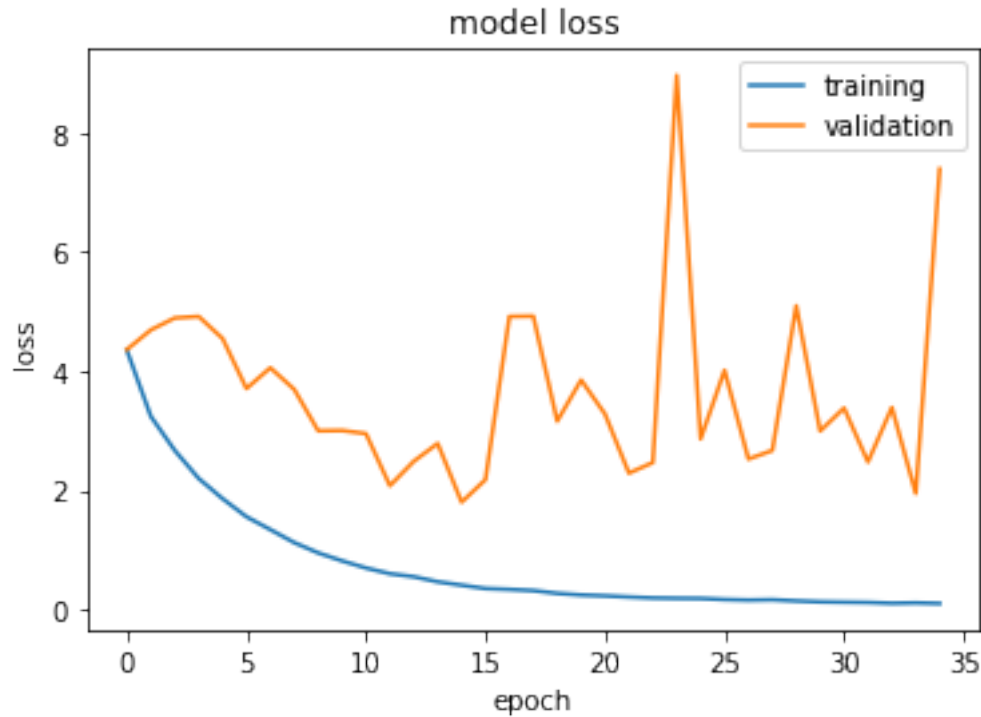
Epoch 23/400  
62/62 [=====] - 31s 505ms/step - loss: 0.2160 - accuracy: 0.9587 - val\_loss: 2.4635 - val\_accuracy: 0.4375  
Epoch 24/400  
62/62 [=====] - 31s 507ms/step - loss: 0.1797 - accuracy: 0.9628 - val\_loss: 8.9669 - val\_accuracy: 0.2229  
Epoch 25/400  
62/62 [=====] - 31s 502ms/step - loss: 0.1637 - accuracy: 0.9541 - val\_loss: 2.8599 - val\_accuracy: 0.3958  
Epoch 26/400  
62/62 [=====] - 31s 503ms/step - loss: 0.1494 - accuracy: 0.9656 - val\_loss: 4.0128 - val\_accuracy: 0.3187  
Epoch 27/400  
62/62 [=====] - 31s 505ms/step - loss: 0.1535 - accuracy: 0.9597 - val\_loss: 2.5226 - val\_accuracy: 0.4479  
Epoch 28/400  
62/62 [=====] - 32s 512ms/step - loss: 0.1725 - accuracy: 0.9620 - val\_loss: 2.6662 - val\_accuracy: 0.4167  
Epoch 29/400  
62/62 [=====] - 32s 524ms/step - loss: 0.1454 - accuracy: 0.9656 - val\_loss: 5.0952 - val\_accuracy: 0.2896  
Epoch 30/400  
62/62 [=====] - 32s 519ms/step - loss: 0.1288 - accuracy: 0.9682 - val\_loss: 2.9926 - val\_accuracy: 0.4271  
Epoch 31/400  
62/62 [=====] - 31s 504ms/step - loss: 0.1090 - accuracy: 0.9715 - val\_loss: 3.3791 - val\_accuracy: 0.3771  
Epoch 32/400  
62/62 [=====] - 32s 518ms/step - loss: 0.1163 - accuracy: 0.9706 - val\_loss: 2.4737 - val\_accuracy: 0.4875  
Epoch 33/400  
62/62 [=====] - 32s 517ms/step - loss: 0.0936 - accuracy: 0.9808 - val\_loss: 3.3878 - val\_accuracy: 0.3688  
Epoch 34/400  
62/62 [=====] - 31s 510ms/step - loss: 0.1080 - accuracy: 0.9750 - val\_loss: 1.9445 - val\_accuracy: 0.5521  
Epoch 35/400  
62/62 [=====] - 32s 510ms/step - loss: 0.1041 - accuracy: 0.9722 - val\_loss: 7.3934 - val\_accuracy: 0.1771  
Restoring model weights from the end of the best epoch.  
Epoch 00035: early stopping

```
[ ]: plt.plot(history6_1.history['accuracy'])
plt.plot(history6_1.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
```

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



```
[ ]: plt.plot(history6_1.history['loss'])
plt.plot(history6_1.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```



#### Architecture from Experiment 4

```
[ ]: model6_2 = Sequential()

# 1st Convolution Layer
model6_2.add(Conv2D(32, input_shape=(*IMAGE_SIZE, 1), kernel_size=3,
    ↪activation=mish))
model6_2.add(BatchNormalization())
model6_2.add(Conv2D(32, kernel_size=3, activation=mish))
model6_2.add(BatchNormalization())
model6_2.add(Conv2D(32, kernel_size=5, strides=2, padding='same',
    ↪activation=mish))
model6_2.add(BatchNormalization())
model6_2.add(Dropout(0.4))

# 2nd Convolution Layer
model6_2.add(Conv2D(64, kernel_size=3, activation=mish))
model6_2.add(BatchNormalization())
model6_2.add(Conv2D(64, kernel_size=3, activation=mish))
model6_2.add(BatchNormalization())
model6_2.add(Conv2D(64, kernel_size=5, strides=2, padding='same',
    ↪activation=mish))
model6_2.add(BatchNormalization())
```

```

model6_2.add(Dropout(0.4))

# 3rd Convolution Layer
model6_2.add(Conv2D(128, kernel_size = 4, activation=mish))
model6_2.add(BatchNormalization())

# Passing to a Fully Connected Layer
model6_2.add(Flatten())
model6_2.add(Dropout(0.4))

# Output Layer
model6_2.add(Dense(62, activation='softmax'))

```

```
[ ]: model6_2.summary()
```

Model: "sequential\_3"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 43, 58, 32)	320
batch_normalization_12 (Batch Normalization)	(None, 43, 58, 32)	128
conv2d_7 (Conv2D)	(None, 41, 56, 32)	9248
batch_normalization_13 (Batch Normalization)	(None, 41, 56, 32)	128
conv2d_8 (Conv2D)	(None, 21, 28, 32)	25632
batch_normalization_14 (Batch Normalization)	(None, 21, 28, 32)	128
dropout_6 (Dropout)	(None, 21, 28, 32)	0
conv2d_9 (Conv2D)	(None, 19, 26, 64)	18496
batch_normalization_15 (Batch Normalization)	(None, 19, 26, 64)	256
conv2d_10 (Conv2D)	(None, 17, 24, 64)	36928
batch_normalization_16 (Batch Normalization)	(None, 17, 24, 64)	256
conv2d_11 (Conv2D)	(None, 9, 12, 64)	102464
batch_normalization_17 (Batch Normalization)	(None, 9, 12, 64)	256
dropout_7 (Dropout)	(None, 9, 12, 64)	0

conv2d_12 (Conv2D)	(None, 6, 9, 128)	131200
-----		
batch_normalization_18 (Batch Normalization)	(None, 6, 9, 128)	512
-----		
flatten_3 (Flatten)	(None, 6912)	0
-----		
dropout_8 (Dropout)	(None, 6912)	0
-----		
dense_9 (Dense)	(None, 62)	428606
=====		
Total params: 754,558		
Trainable params: 753,726		
Non-trainable params: 832		
-----		

```
[ ]: model6_2.compile(loss='categorical_crossentropy', optimizer=Adam(),
    ↪metrics=['accuracy'])
```

Saving the Model Checkpoint

```
[ ]: checkpoint_filepath6_2 = 'exp6_2/checkpoint'
model_checkpoint_callback6_2 = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath6_2,
    save_weights_only=True,
    monitor='val_loss',
    mode='min',
    save_best_only=True)
```

```
[ ]: history6_2 = model6_2.fit(
    train_generator3,
    epochs=EPOCHS,
    validation_data=validation_generator3,
    steps_per_epoch = train_generator3.samples // BATCH_SIZE,
    validation_steps = validation_generator3.samples // BATCH_SIZE,
    callbacks=[model_checkpoint_callback6_2, early_stopping_callback]
)
```

Epoch 1/400

62/62 [=====] - 33s 506ms/step - loss: 5.2828 - accuracy: 0.0535 - val\_loss: 5.1547 - val\_accuracy: 0.0167

Epoch 2/400

62/62 [=====] - 31s 499ms/step - loss: 2.6122 - accuracy: 0.3701 - val\_loss: 6.4788 - val\_accuracy: 0.0250

Epoch 3/400

62/62 [=====] - 31s 499ms/step - loss: 1.6781 - accuracy: 0.5677 - val\_loss: 6.7860 - val\_accuracy: 0.0500

Epoch 4/400

62/62 [=====] - 31s 498ms/step - loss: 1.2515 -

accuracy: 0.6698 - val\_loss: 7.6091 - val\_accuracy: 0.0625  
 Epoch 5/400  
 62/62 [=====] - 31s 501ms/step - loss: 1.0160 -  
 accuracy: 0.7232 - val\_loss: 11.6267 - val\_accuracy: 0.0479  
 Epoch 6/400  
 62/62 [=====] - 31s 498ms/step - loss: 0.7533 -  
 accuracy: 0.7794 - val\_loss: 6.4720 - val\_accuracy: 0.1646  
 Epoch 7/400  
 62/62 [=====] - 31s 497ms/step - loss: 0.5145 -  
 accuracy: 0.8392 - val\_loss: 14.5081 - val\_accuracy: 0.0417  
 Epoch 8/400  
 62/62 [=====] - 31s 499ms/step - loss: 0.4077 -  
 accuracy: 0.8741 - val\_loss: 7.0863 - val\_accuracy: 0.1396  
 Epoch 9/400  
 62/62 [=====] - 31s 498ms/step - loss: 0.3797 -  
 accuracy: 0.8741 - val\_loss: 4.7880 - val\_accuracy: 0.3396  
 Epoch 10/400  
 62/62 [=====] - 31s 497ms/step - loss: 0.3910 -  
 accuracy: 0.8966 - val\_loss: 2.4809 - val\_accuracy: 0.5125  
 Epoch 11/400  
 62/62 [=====] - 31s 500ms/step - loss: 0.2723 -  
 accuracy: 0.9176 - val\_loss: 3.2420 - val\_accuracy: 0.4375  
 Epoch 12/400  
 62/62 [=====] - 31s 496ms/step - loss: 0.2342 -  
 accuracy: 0.9413 - val\_loss: 4.5787 - val\_accuracy: 0.3562  
 Epoch 13/400  
 62/62 [=====] - 31s 498ms/step - loss: 0.2054 -  
 accuracy: 0.9412 - val\_loss: 2.6093 - val\_accuracy: 0.5813  
 Epoch 14/400  
 62/62 [=====] - 31s 495ms/step - loss: 0.2238 -  
 accuracy: 0.9324 - val\_loss: 4.7339 - val\_accuracy: 0.4062  
 Epoch 15/400  
 62/62 [=====] - 30s 493ms/step - loss: 0.2045 -  
 accuracy: 0.9379 - val\_loss: 8.2359 - val\_accuracy: 0.2833  
 Epoch 16/400  
 62/62 [=====] - 31s 495ms/step - loss: 0.1736 -  
 accuracy: 0.9482 - val\_loss: 4.0421 - val\_accuracy: 0.4083  
 Epoch 17/400  
 62/62 [=====] - 31s 493ms/step - loss: 0.2080 -  
 accuracy: 0.9468 - val\_loss: 5.7639 - val\_accuracy: 0.3083  
 Epoch 18/400  
 62/62 [=====] - 31s 493ms/step - loss: 0.1215 -  
 accuracy: 0.9621 - val\_loss: 7.6276 - val\_accuracy: 0.2583  
 Epoch 19/400  
 62/62 [=====] - 30s 491ms/step - loss: 0.1403 -  
 accuracy: 0.9562 - val\_loss: 6.9810 - val\_accuracy: 0.2542  
 Epoch 20/400  
 62/62 [=====] - 31s 495ms/step - loss: 0.1488 -

accuracy: 0.9545 - val\_loss: 2.9153 - val\_accuracy: 0.5292  
 Epoch 21/400  
 62/62 [=====] - 32s 518ms/step - loss: 0.1313 -  
 accuracy: 0.9593 - val\_loss: 2.3372 - val\_accuracy: 0.5958  
 Epoch 22/400  
 62/62 [=====] - 32s 521ms/step - loss: 0.1090 -  
 accuracy: 0.9694 - val\_loss: 4.9452 - val\_accuracy: 0.4042  
 Epoch 23/400  
 62/62 [=====] - 32s 511ms/step - loss: 0.1232 -  
 accuracy: 0.9624 - val\_loss: 11.5529 - val\_accuracy: 0.1604  
 Epoch 24/400  
 62/62 [=====] - 32s 514ms/step - loss: 0.0993 -  
 accuracy: 0.9700 - val\_loss: 3.2483 - val\_accuracy: 0.5583  
 Epoch 25/400  
 62/62 [=====] - 31s 497ms/step - loss: 0.1000 -  
 accuracy: 0.9740 - val\_loss: 2.8103 - val\_accuracy: 0.5625  
 Epoch 26/400  
 62/62 [=====] - 31s 494ms/step - loss: 0.1198 -  
 accuracy: 0.9633 - val\_loss: 4.5392 - val\_accuracy: 0.4792  
 Epoch 27/400  
 62/62 [=====] - 31s 498ms/step - loss: 0.1238 -  
 accuracy: 0.9620 - val\_loss: 9.9924 - val\_accuracy: 0.2208  
 Epoch 28/400  
 62/62 [=====] - 31s 494ms/step - loss: 0.1024 -  
 accuracy: 0.9735 - val\_loss: 2.9984 - val\_accuracy: 0.5792  
 Epoch 29/400  
 62/62 [=====] - 31s 498ms/step - loss: 0.0960 -  
 accuracy: 0.9717 - val\_loss: 4.3783 - val\_accuracy: 0.4750  
 Epoch 30/400  
 62/62 [=====] - 31s 496ms/step - loss: 0.1184 -  
 accuracy: 0.9671 - val\_loss: 6.5769 - val\_accuracy: 0.3479  
 Epoch 31/400  
 62/62 [=====] - 31s 503ms/step - loss: 0.1328 -  
 accuracy: 0.9623 - val\_loss: 2.7831 - val\_accuracy: 0.5938  
 Epoch 32/400  
 62/62 [=====] - 31s 502ms/step - loss: 0.1031 -  
 accuracy: 0.9680 - val\_loss: 12.7633 - val\_accuracy: 0.1708  
 Epoch 33/400  
 62/62 [=====] - 31s 497ms/step - loss: 0.1445 -  
 accuracy: 0.9608 - val\_loss: 2.7398 - val\_accuracy: 0.5938  
 Epoch 34/400  
 62/62 [=====] - 31s 498ms/step - loss: 0.0686 -  
 accuracy: 0.9757 - val\_loss: 2.6970 - val\_accuracy: 0.6313  
 Epoch 35/400  
 62/62 [=====] - 31s 496ms/step - loss: 0.0559 -  
 accuracy: 0.9780 - val\_loss: 3.2838 - val\_accuracy: 0.5396  
 Epoch 36/400  
 62/62 [=====] - 31s 498ms/step - loss: 0.0626 -



```

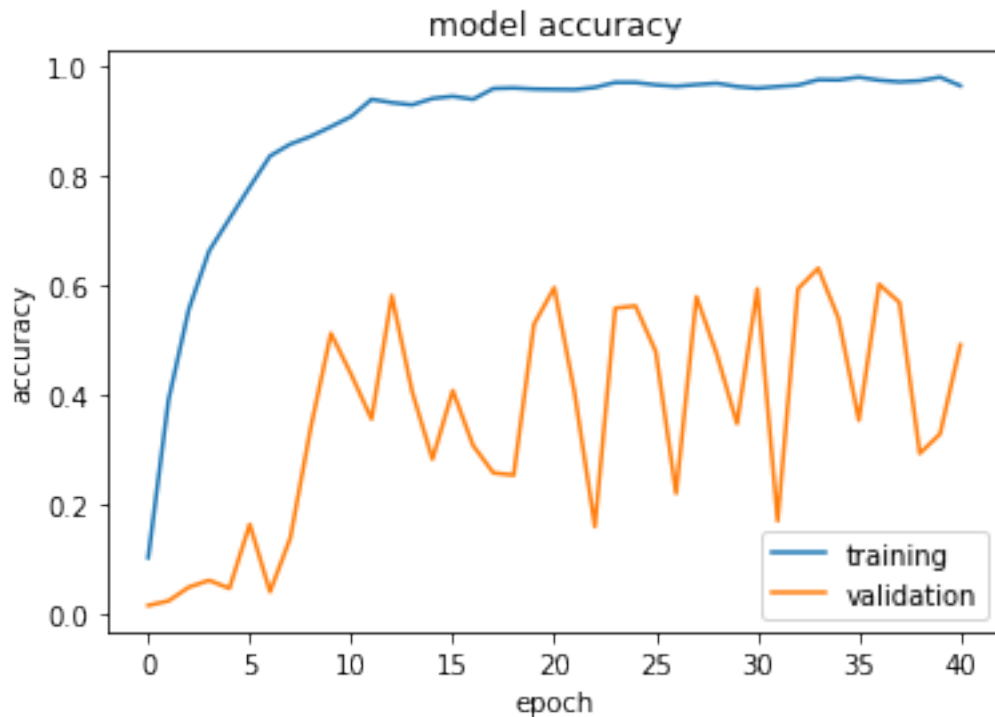
accuracy: 0.9802 - val_loss: 7.0936 - val_accuracy: 0.3542
Epoch 37/400
62/62 [=====] - 31s 504ms/step - loss: 0.0741 -
accuracy: 0.9786 - val_loss: 2.8313 - val_accuracy: 0.6021
Epoch 38/400
62/62 [=====] - 31s 500ms/step - loss: 0.0796 -
accuracy: 0.9741 - val_loss: 3.2906 - val_accuracy: 0.5688
Epoch 39/400
62/62 [=====] - 31s 500ms/step - loss: 0.0906 -
accuracy: 0.9743 - val_loss: 10.2623 - val_accuracy: 0.2937
Epoch 40/400
62/62 [=====] - 31s 502ms/step - loss: 0.0475 -
accuracy: 0.9831 - val_loss: 8.2991 - val_accuracy: 0.3292
Epoch 41/400
62/62 [=====] - 31s 504ms/step - loss: 0.0958 -
accuracy: 0.9667 - val_loss: 4.2879 - val_accuracy: 0.4917
Restoring model weights from the end of the best epoch.
Epoch 00041: early stopping

```

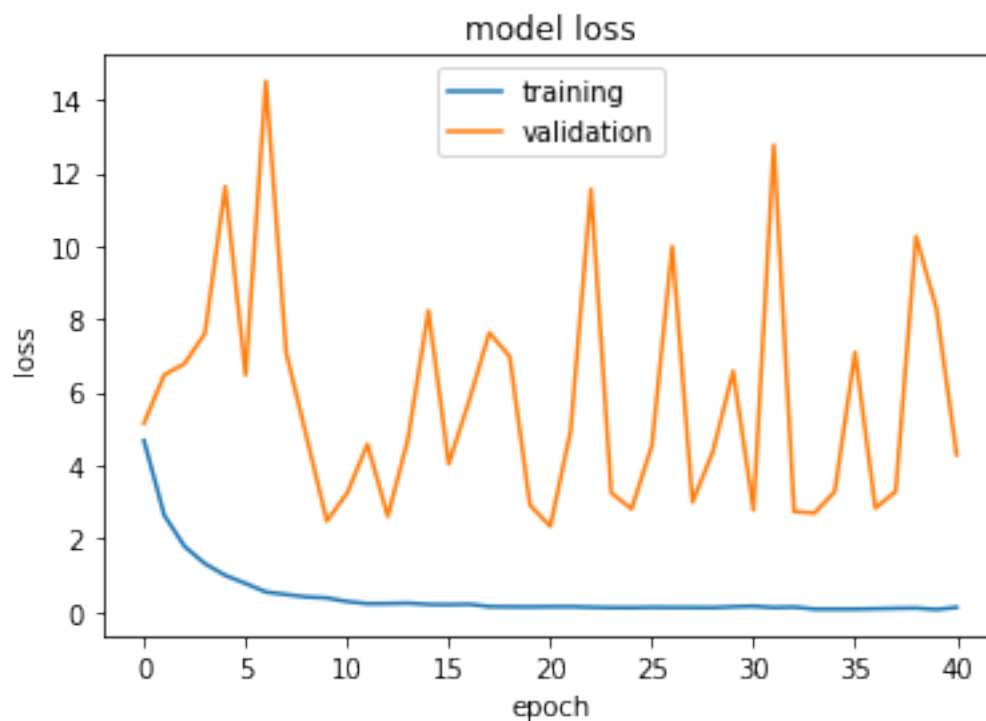
```

[ ]: plt.plot(history6_2.history['accuracy'])
plt.plot(history6_2.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()

```



```
[ ]: plt.plot(history6_2.history['loss'])
plt.plot(history6_2.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='best')
plt.show()
```



We see that batch size of 32 actually performs worse than that of 64.

#### 1.4 Observations form these experiments

- I downscaled the images by 20x while making sure that the images are still recognizable from each other. This would reduce the number of parameters in our network.
- I started with a modified version of the original LeNet, which was used to classify on the MNIST dataset, as our current dataset closely resembles it, I got an accuracy of a little below 60%.
- Data Augmentation did not help for this dataset as the augmented data was very different from the original data and it actually performed worse. I didn not use Data augmentation for later experiments.
- I tried the Mish Activation function instead of ReLU and it reached the the same accuracy in lesser epochs, however it had some variance issues for the training accuracy in the later

epochs.

- I tried different architectures: One which is known to give pretty good accuracy on the MNIST dataset in Kaggle Competitions and one Modified EfficientNet Architecture.
  - The first of them gave similar result to the original LeNet and in lesser epochs, but had larger variance in training accuracy in the later epochs.
  - The EfficientNet architecture actually gave the worst results in all of the experiments, but I think this was due to the scarcity of training data and not using a pretrained model.
- I tried reducing Batch size from 64 to 32, but it gave worse results than 64.
- Lastly, in experiment 7, (which I actually added in the section of experiment 3), I used higher temperature to achieve ‘softer’ softmax outputs, alongwith the Mish Activation and it gave the best accuracy on the validations et so far.
- The best validation accuracy that I achieved was 59.15% in Experiment 7.

## 1.5 Conclusions and Future Improvements

- Overall, I think the original LeNet with Mish activation and high temperature softmax from Experiment 7 and the Model Architecture from Experiment 4, both gave good results. Clearly, the models were overfitting the data and I think that if trained with more data, they would perform better. For the future parts of this task, I would be using these two architectures, and would even like to try high temperature softmax for Experiment 4’s architecture.
- EfficientNet can give better results than from Experiment 5 but again, it needs more data and some pre-trained weights. EfficientNetV2, released recently can also be a good model, but I can’t find an Open Source Implementation of it and it’s a little complex to build it in this duration, so I didn’t try it, but I’m sure that it would give better results than what I got from EfficientNet here.

## 2 References

- [1] [EMNIST handwritten character recognition with Deep Learning](#)
- [2] [How to choose CNN Architecture MNIST](#)
- [3] [Swish Vs Mish: Latest Activation Functions](#)
- [4] [Mish Class Definition in Keras](#)
- [5] [Mnist\\_EfficientNet Kaggle Notebook](#)
- [6] [Image classification via fine-tuning with EfficientNet](#)
- [7] [Keras EfficientNet Implementation Source Code](#)
- [8] [Keras ModelCheckpoint Documentaion](#)
- [9] [Keras EarlyStopping Documentaion](#)
- [10] [Keras ImageDataGenerator Documentation](#)
- [11] [Keras EfficientNetB0 Documentation](#)
- [12] [Effect of Batch Size on Neural Net Training](#)
- [13] [How does temperature affect softmax in machine learning?](#)
- [14] [Temperatures softmax · Issue #3092 · keras-team/keras](#)
- [15] [EfficientNetV2: Smaller Models and Faster Training](#)

[ ]: