z_jainayushri_179292_59921786_Jain_Ayushri_Project_1_Deep_Learning

October 13, 2023

1 Utilities

1.1 Import necessary libraries

[ ]:
```
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.callbacks import EarlyStopping, LambdaCallback
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Dense,
 ↪Flatten, Dropout, Activation, BatchNormalization
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.regularizers import l2, l1_l2
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras import optimizers
from keras.layers import GaussianNoise
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import tensorflow as tf
import pandas as pd
import math
from matplotlib import pyplot as plt
import os
import pickle
import random
from tensorflow.python.ops.numpy_ops import np_config
np_config.enable_numpy_behavior()
```

1.2 Load dataset

I uploaded the dataset to drive. So retrieving the data set from drive.

```
[ ]: from google.colab import drive
drive.mount('/content/drive')
os.listdir('drive/MyDrive/DeepLearningCSCE636')

csce636_train_images = pickle.load(open('drive/MyDrive/DeepLearningCSCE636/
  ↪636_project1_train_images', 'rb'))
csce636_train_labels = pickle.load(open('drive/MyDrive/DeepLearningCSCE636/
  ↪636_project1_train_labels', 'rb'))

csce636_train_images = csce636_train_images.reshape((csce636_train_images.
  ↪shape[0], 28, 28, 1))
csce636_train_images = csce636_train_images.astype('float') / 255.
```

```
Mounted at /content/drive
```

Create the training and test set from the given dataset using 20:80 split.

```
[ ]: train_images, test_images, train_labels, test_labels =␣
  ↪train_test_split(csce636_train_images, csce636_train_labels, test_size = 0.
  ↪2, random_state=42)
```

```
[ ]: # renaming the variables for simplicity
X_train = train_images
y_train = train_labels
X_test = test_images
y_test = test_labels
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(48000, 28, 28, 1)
(48000,)
(12000, 28, 28, 1)
(12000,)
```

1.3 Function to plot training and validation accuracy and loss

```
[ ]: import matplotlib.pyplot as plt
def plot_and_print(history, model, X_test, y_test):
    # Set the figure size for both plots
    plt.figure(figsize=(10, 10))  # Adjust the width and height as needed
```

```
    acc = history.history["accuracy"]
    val_acc = history.history["val_accuracy"]
    loss = history.history["loss"]
    val_loss = history.history["val_loss"]
    epochs = range(1, len(acc) + 1)

    # Plot training and validation accuracy
    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, "bo", label="Training accuracy")
    plt.plot(epochs, val_acc, "b", label="Validation accuracy")
    plt.title("Training and validation accuracy")
    plt.legend()

    # Plot training and validation loss
    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, "bo", label="Training loss")
    plt.plot(epochs, val_loss, "b", label="Validation loss")
    plt.title("Training and validation loss")
    plt.legend()

    # Adjust the layout and display the plot
    plt.tight_layout()
    plt.show()

    # Evaluation on the test set
    test_loss, test_acc = model.evaluate(X_test, y_test)
    print(f"Test accuracy: {test_acc:.3f}")
```

1.4 Define callback functions

[ ]:
```
# using variable learning rate for all models
variable_learning_rate = ReduceLROnPlateau(monitor='val_loss', factor = 0.1,␣
 ↪patience = 5) #, min_lr = 0.0000001)

# using early stopping if validation loss does not improve after 10 epochs
early_stopping = keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

# using checpointing to save the best trained weights
model_checkpoint = keras.callbacks.ModelCheckpoint(
    filepath = 'csce636_project_1.x',
    save_best_only = True,
    monitor = 'val_loss')
```

1.5 Function to create and train model: common code for multiple models

[ ]:
```
def create_and_train_model(inputs, x, X_train = X_train, y_train = y_train,␣
↪X_test = X_test, y_test = y_test):
    # define output layer
    outputs = layers.Dense(10, activation="softmax")(x)
    # define model
    model = keras.Model(inputs=inputs, outputs=outputs)
    # compile model
    model.compile(optimizer = optimizers.RMSprop(),
      loss="sparse_categorical_crossentropy",
      metrics=["accuracy"])
    # train model using cross validation split of 20% for 100 epochs
    history = model.fit(X_train,
                        y_train,
                        epochs=100,
                        validation_split = 0.2,
                        callbacks=[early_stopping, variable_learning_rate,␣
↪model_checkpoint],
                        batch_size=256,
                        verbose=False)
    plot_and_print(history, model, X_test, y_test)
    return model
```

1.6 Define input layer

[ ]:
```
# defining inputs (common for model)
inputs = keras.Input(shape=(28, 28, 1))
```

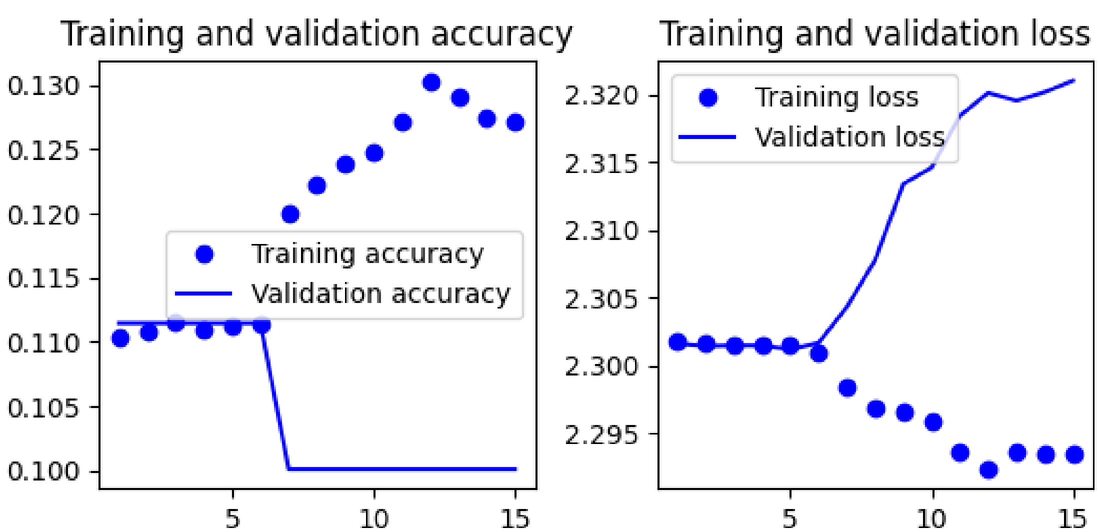
2 Creating different models

2.1 Set 1 : Use multiple convolutional, maxpooling and dropout layers

Keep same padding for all layers. Keep filter size from 4 to 512 in convolutional layers. Gives 11% accuracy

[ ]:
```
# create layers for the model
x = layers.Conv2D(filters=4, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=8, kernel_size=3, activation="relu",␣
↪padding='same')(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
```

```
x = layers.Conv2D(filters=16, kernel_size=3, activation="relu",␣
 ↪padding='same')(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu",␣
 ↪padding='same')(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu",␣
 ↪padding='same')(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu",␣
 ↪padding='same')(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu",␣
 ↪padding='same')(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=512, kernel_size=3, activation="relu",␣
 ↪padding='same')(x)
x = layers.Flatten()(x)

# create and train model
model = create_and_train_model(inputs, x)
```

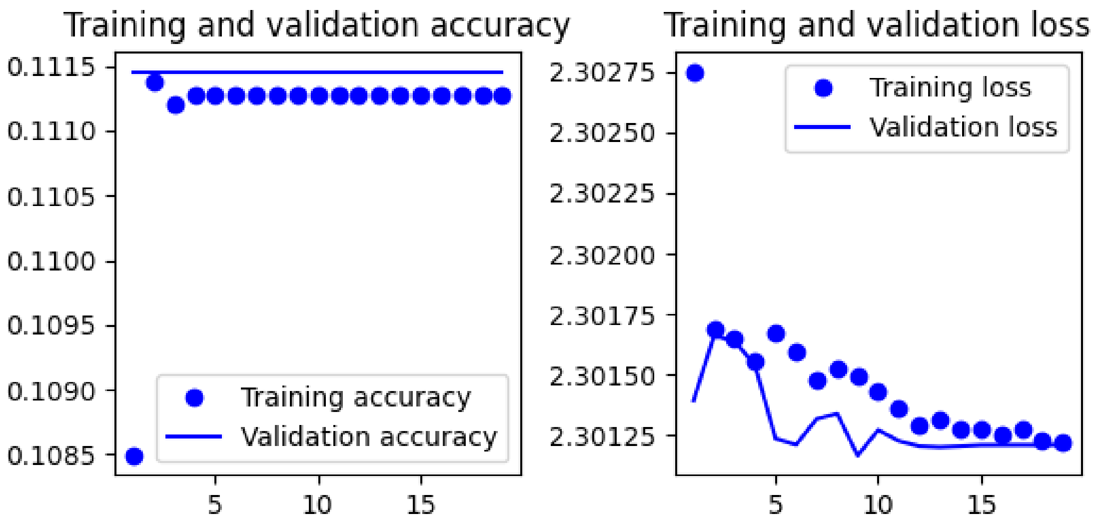



```
375/375 [==============================] - 1s 3ms/step - loss: 2.3013 -
accuracy: 0.1117
```

```
Test accuracy: 0.112
```

Keep same padding for all layers. Keep filter size from 16 to 512 in convolutional layers. Gives 11% accuracy

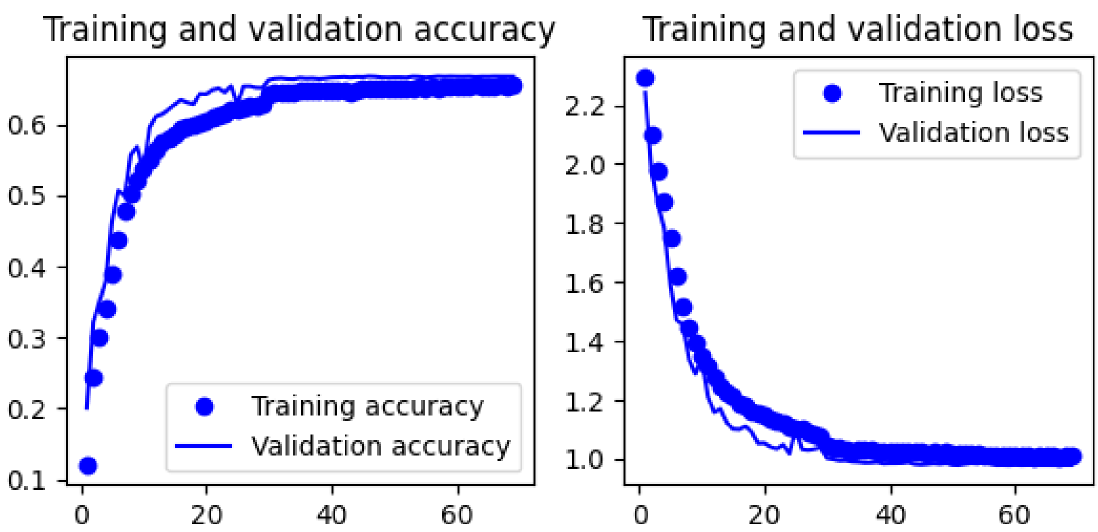
```
[ ]: # create layers for the model
x = layers.Conv2D(filters=16, kernel_size=3, activation="relu",␣
 ↪padding='same')(inputs)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu",␣
 ↪padding='same')(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu",␣
 ↪padding='same')(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu",␣
 ↪padding='same')(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu",␣
 ↪padding='same')(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=512, kernel_size=3, activation="relu",␣
 ↪padding='same')(x)
x = layers.Flatten()(x)

# create and train model
model = create_and_train_model(inputs, x)
```




```
375/375 [==============================] - 2s 4ms/step - loss: 2.3013 -
accuracy: 0.1117
Test accuracy: 0.112
```

Remove padding for convolutional layers. Keep filter size from 32 to 128 in convolutional layers. Reduce number of layers. Gives 66.6% accuracy

```
[ ]: # create layers for the model
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)

# create and train model
model = create_and_train_model(inputs, x)
```



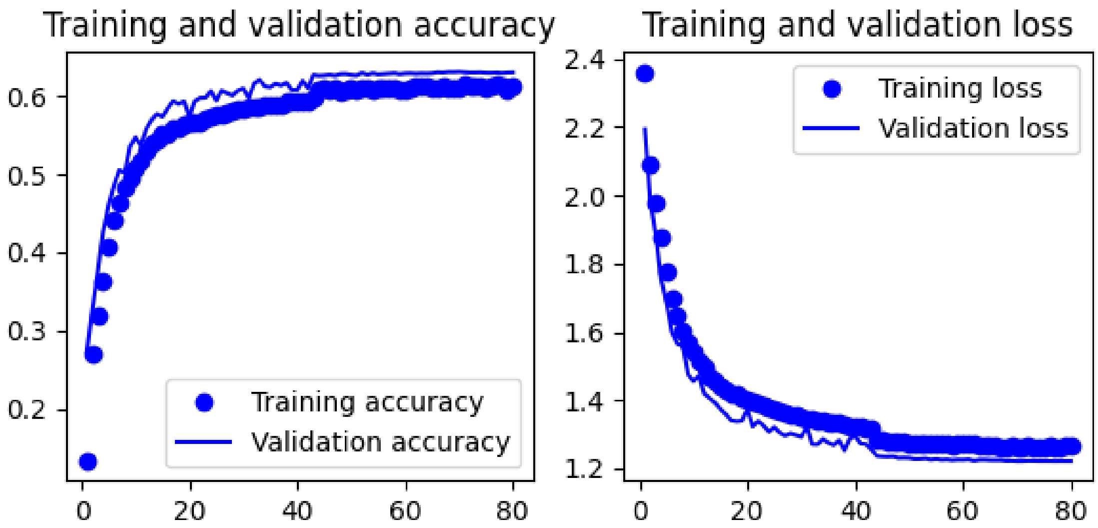

```
375/375 [==============================] - 1s 3ms/step - loss: 0.9828 -
accuracy: 0.6659
Test accuracy: 0.666
```

Try fine tuning

- Remove padding for convolutional layers
- Keep filter size from 32 to 128 in convolutional layers
- Change dropout percentage
- Add regularization

Gives 62.8% accuracy

```
[ ]: # create layers for the model
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu",␣
 ↪kernel_regularizer=l2(0.001))(inputs)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu",␣
 ↪kernel_regularizer=l2(0.001))(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu",␣
 ↪kernel_regularizer=l2(0.001))(x)
x = layers.Flatten()(x)

# create and train model
model = create_and_train_model(inputs, x)
```

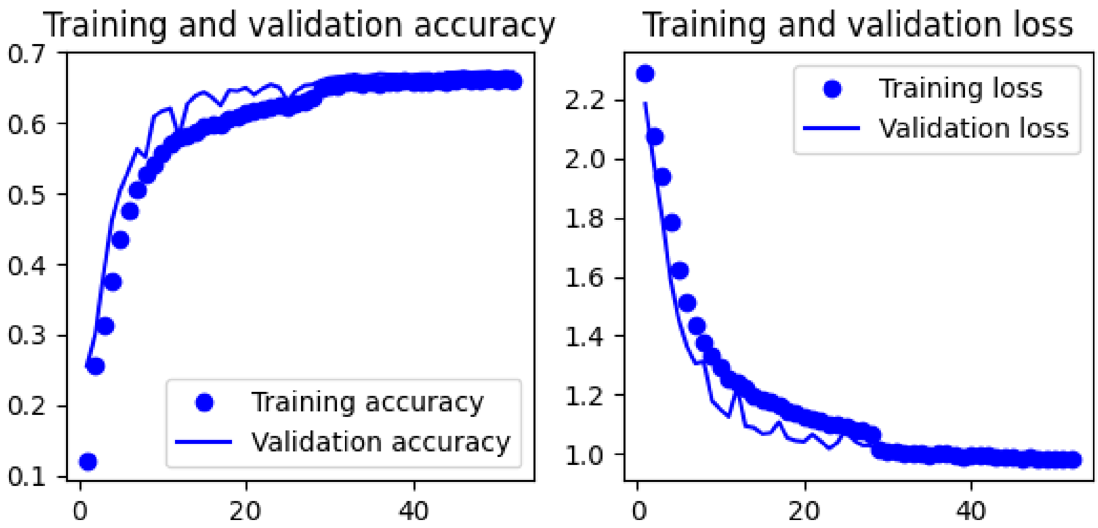



```
375/375 [==============================] - 1s 4ms/step - loss: 1.2242 -
accuracy: 0.6285
Test accuracy: 0.628
```

2.2 Set 2: Use convolutional, maxpooling, dense and dropout layers.

Remove padding for convolutional layers. Keep filter size from 32 to 128 in convolutional layers. Add a dense layer. Gives 66.3% accuracy

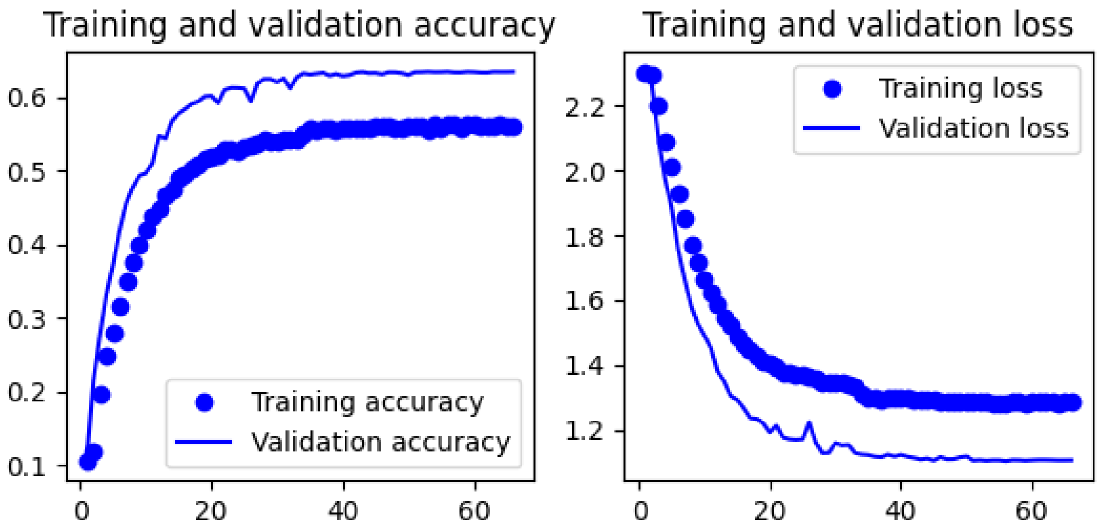
```
[ ]: # create layers for the model
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dense(units = 128, use_bias=True)(x)
# create and train model
model = create_and_train_model(inputs, x)
```




```
375/375 [==============================] - 1s 4ms/step - loss: 0.9828 -
accuracy: 0.6633
Test accuracy: 0.663
```

Remove padding for convolutional layers. Keep filter size from 16 to 64 in convolutional layers. Add a dense layer Gives 62.8% accuracy

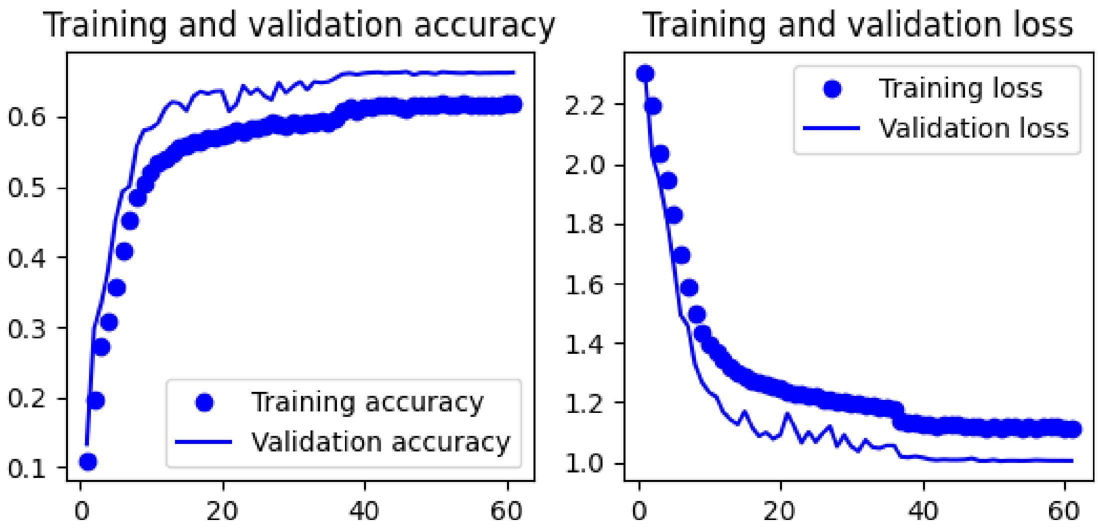
[ ]:
```
# create layers for the model
x = layers.Conv2D(filters=16, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.Dropout(0.5)(x)
x = layers.Flatten()(x)
x = layers.Dense(units = 128, use_bias=True)(x)
# create and train model
model = create_and_train_model(inputs, x)
```




```
375/375 [==============================] - 1s 3ms/step - loss: 1.1083 -
accuracy: 0.6283
Test accuracy: 0.628
```

Remove padding for convolutional layers. Keep filter size from 16 to 64 in convolutional layers. Add 2 dense layers Gives 65.9% accuracy

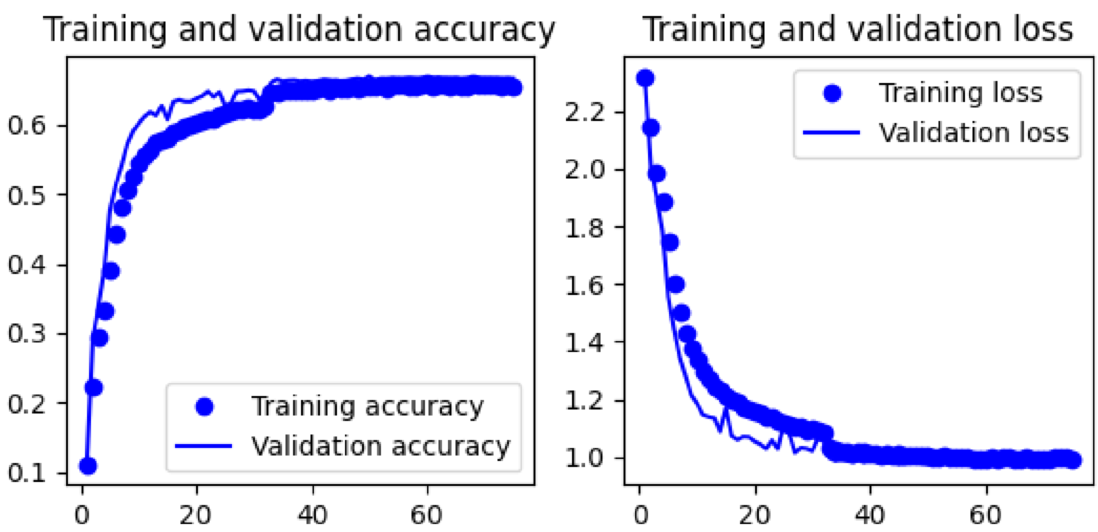
```
[ ]: # create layers for the model
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Dropout(0.5)(x)
x = layers.Flatten()(x)
x = layers.Dense(units = 64, use_bias=True)(x)
x = layers.Dense(units = 32, use_bias=True)(x)
# create and train model
model = create_and_train_model(inputs, x)
```




```
375/375 [==============================] - 1s 3ms/step - loss: 1.0090 -
accuracy: 0.6590
Test accuracy: 0.659
```

Increase size of dense layer = 256   Gives 66.5% accuracy

[ ]:
```
# create layers for the model
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2, padding='same')(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dense(units = 256, use_bias=True)(x)
# create and train model
model = create_and_train_model(inputs, x)
```



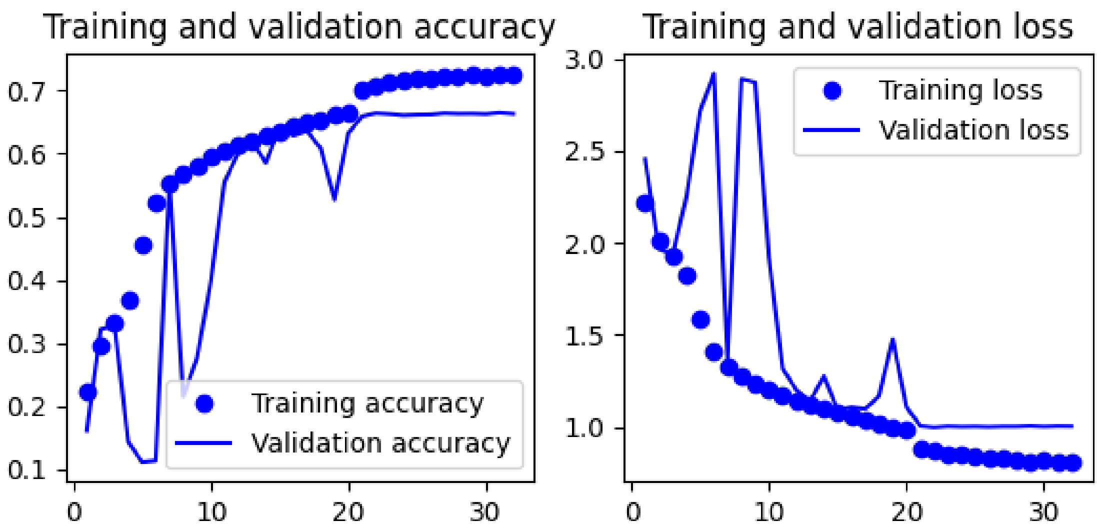

```
375/375 [==============================] - 1s 3ms/step - loss: 0.9823 -
accuracy: 0.6648
Test accuracy: 0.665
```

2.3 Set 3: Inspiration from this article.

Keeping model as it is in the article Gives 65.9% accuracy

[ ]:
```
x = layers.Conv2D(filters = 32, kernel_size = 5, strides = 1, activation =␣
↪'relu', input_shape = (32,32,1), kernel_regularizer=l2(0.0005))(inputs)
x = layers.Conv2D(filters = 32, kernel_size = 5, strides = 1, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size = 2, strides = 2)(x)
x = layers.Dropout(0.25)(x)
x = layers.Conv2D(filters = 64, kernel_size = 3, strides = 1, activation =␣
↪'relu', kernel_regularizer=l2(0.0005))(x)
x = layers.Conv2D(filters = 64, kernel_size = 3, strides = 1, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size = 2, strides = 2)(x)
x = layers.Dropout(0.25)(x)
x = layers.Flatten()(x)
x = layers.Dense(units = 256, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dense(units = 128, use_bias=False)(x)
x = layers.BatchNormalization()(x)
```

```
x = layers.Activation('relu')(x)
x = layers.Dense(units = 84, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dropout(0.25)(x)
model = create_and_train_model(inputs, x)
```




```
375/375 [==============================] - 2s 5ms/step - loss: 1.0178 -
accuracy: 0.6588
Test accuracy: 0.659
```

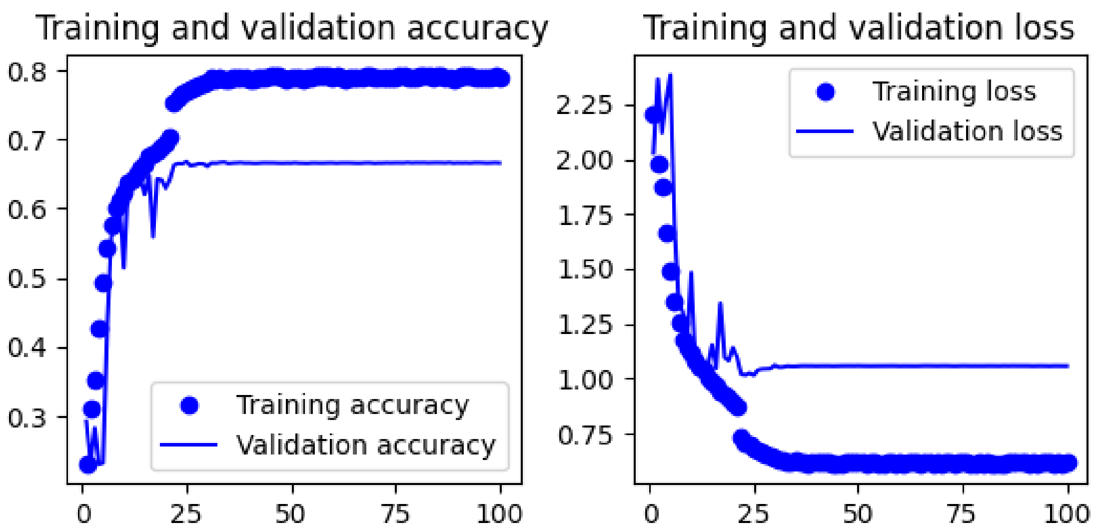
Adding Preprocessing Step

[ ]:
```
new_X_train = np.array(X_train)
new_X_test = np.array(X_test)

# Padding the images by 2 pixels since in the paper input images were 32x32
new_X_train = np.pad(new_X_train, ((0,0),(2,2),(2,2),(0,0)), 'constant')
new_X_test = np.pad(new_X_test, ((0,0),(2,2),(2,2),(0,0)), 'constant')

# Standardization
mean_px = new_X_train.mean().astype(np.float32)
std_px = new_X_train.std().astype(np.float32)
new_X_train = (new_X_train - mean_px)/(std_px)

mean_px = new_X_test.mean().astype(np.float32)
std_px = new_X_test.std().astype(np.float32)
new_X_test = (new_X_test - mean_px)/(std_px)
```

```
# One-hot encoding the labels
new_y_train = to_categorical(y_train, num_classes = 10)
new_y_test = to_categorical(y_test, num_classes = 10)
```

```
[ ]: new_inputs = keras.Input(shape=(32, 32, 1))
x = layers.Conv2D(filters = 32, kernel_size = 5, strides = 1, activation =␣
↪'relu', input_shape = (32,32,1), kernel_regularizer=l2(0.0005))(new_inputs)
x = layers.Conv2D(filters = 32, kernel_size = 5, strides = 1, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size = 2, strides = 2)(x)
x = layers.Dropout(0.25)(x)
x = layers.Conv2D(filters = 64, kernel_size = 3, strides = 1, activation =␣
↪'relu', kernel_regularizer=l2(0.0005))(x)
x = layers.Conv2D(filters = 64, kernel_size = 3, strides = 1, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size = 2, strides = 2)(x)
x = layers.Dropout(0.25)(x)
x = layers.Flatten()(x)
x = layers.Dense(units = 256, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dense(units = 128, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dense(units = 84, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dropout(0.25)(x)
# model = create_and_train_model(new_inputs, x, new_X_train, new_y_train,␣
↪new_X_test, new_y_test)
# define output layer
new_outputs = layers.Dense(10, activation="softmax")(x)
# define model
model1 = keras.Model(inputs=new_inputs, outputs=new_outputs)
# compile model
model1.compile(optimizer = optimizers.RMSprop(),
  loss="categorical_crossentropy",
  metrics=["accuracy"])
# train model using validation split of 20% for 100 epochs
history1 = model1.fit(new_X_train,
                    new_y_train,
                    epochs=100,
                    validation_split = 0.2,
                    # callbacks=[early_stopping, variable_learning_rate,␣
↪model_checkpoint],
```

```
                    callbacks=[variable_learning_rate, model_checkpoint],
                    batch_size=256,
                    verbose=False)
plot_and_print(history1, model1, new_X_test, new_y_test)
```




```
375/375 [==============================] - 1s 3ms/step - loss: 1.0793 -
accuracy: 0.6591
Test accuracy: 0.659
```

Updating layers and parameter Gives 69.3% accuracy

```
[ ]: new_inputs = keras.Input(shape=(32, 32, 1))
x = layers.Conv2D(filters = 32, kernel_size = 5, strides = 1, activation =␣
 ↪'relu', input_shape = (32,32,1), kernel_regularizer=l2(0.0005))(new_inputs)
x = layers.Conv2D(filters = 32, kernel_size = 5, strides = 1, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size = 2, strides = 2)(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters = 64, kernel_size = 3, strides = 1, activation =␣
 ↪'relu', kernel_regularizer=l2(0.0005))(x)
x = layers.Conv2D(filters = 64, kernel_size = 3, strides = 1, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size = 2, strides = 2)(x)
x = layers.Dropout(0.5)(x)
x = layers.Flatten()(x)
x = layers.Dense(units = 256, use_bias=False)(x)
```

```
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dense(units = 128, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dense(units = 64, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dropout(0.5)(x)

# define output layer
new_outputs = layers.Dense(10, activation="softmax")(x)
# define model
model1 = keras.Model(inputs=new_inputs, outputs=new_outputs)
# compile model
model1.compile(optimizer = optimizers.Adam(),
  loss="categorical_crossentropy",
  metrics=["accuracy"])
# train model using validation split of 20% for 100 epochs
history1 = model1.fit(new_X_train,
                    new_y_train,
                    epochs=100,
                    validation_split = 0.2,
                    # callbacks=[early_stopping, variable_learning_rate,␣
 ↪model_checkpoint],
                    callbacks=[variable_learning_rate, model_checkpoint],
                    batch_size=64,
                    verbose=True)
plot_and_print(history1, model1, new_X_test, new_y_test)
```

```
Epoch 1/100
600/600 [==============================] - 15s 12ms/step - loss: 2.4091 -
accuracy: 0.1627 - val_loss: 2.0250 - val_accuracy: 0.2903 - lr: 0.0010
Epoch 2/100
600/600 [==============================] - 6s 10ms/step - loss: 2.0894 -
accuracy: 0.2679 - val_loss: 1.9971 - val_accuracy: 0.3013 - lr: 0.0010
Epoch 3/100
600/600 [==============================] - 5s 9ms/step - loss: 2.0229 -
accuracy: 0.2982 - val_loss: 1.9143 - val_accuracy: 0.3397 - lr: 0.0010
Epoch 4/100
600/600 [==============================] - 5s 9ms/step - loss: 1.8733 -
accuracy: 0.3615 - val_loss: 2.0226 - val_accuracy: 0.2974 - lr: 0.0010
Epoch 5/100
600/600 [==============================] - 6s 10ms/step - loss: 1.6006 -
accuracy: 0.4678 - val_loss: 1.3449 - val_accuracy: 0.5616 - lr: 0.0010
Epoch 6/100
600/600 [==============================] - 5s 9ms/step - loss: 1.4505 -
```

```
accuracy: 0.5214 - val_loss: 1.1936 - val_accuracy: 0.6058 - lr: 0.0010
Epoch 7/100
600/600 [==============================] - 6s 10ms/step - loss: 1.3804 -
accuracy: 0.5449 - val_loss: 1.1925 - val_accuracy: 0.6011 - lr: 0.0010
Epoch 8/100
600/600 [==============================] - 5s 9ms/step - loss: 1.3285 -
accuracy: 0.5656 - val_loss: 1.1342 - val_accuracy: 0.6219 - lr: 0.0010
Epoch 9/100
600/600 [==============================] - 6s 9ms/step - loss: 1.3006 -
accuracy: 0.5724 - val_loss: 1.1098 - val_accuracy: 0.6346 - lr: 0.0010
Epoch 10/100
600/600 [==============================] - 6s 10ms/step - loss: 1.2834 -
accuracy: 0.5810 - val_loss: 1.0934 - val_accuracy: 0.6324 - lr: 0.0010
Epoch 11/100
600/600 [==============================] - 5s 9ms/step - loss: 1.2611 -
accuracy: 0.5890 - val_loss: 1.0683 - val_accuracy: 0.6431 - lr: 0.0010
Epoch 12/100
600/600 [==============================] - 6s 10ms/step - loss: 1.2412 -
accuracy: 0.5968 - val_loss: 1.0502 - val_accuracy: 0.6491 - lr: 0.0010
Epoch 13/100
600/600 [==============================] - 5s 9ms/step - loss: 1.2328 -
accuracy: 0.6015 - val_loss: 1.0775 - val_accuracy: 0.6479 - lr: 0.0010
Epoch 14/100
600/600 [==============================] - 6s 10ms/step - loss: 1.2144 -
accuracy: 0.6043 - val_loss: 1.0644 - val_accuracy: 0.6489 - lr: 0.0010
Epoch 15/100
600/600 [==============================] - 6s 10ms/step - loss: 1.2061 -
accuracy: 0.6065 - val_loss: 1.0355 - val_accuracy: 0.6625 - lr: 0.0010
Epoch 16/100
600/600 [==============================] - 5s 9ms/step - loss: 1.2012 -
accuracy: 0.6080 - val_loss: 1.0486 - val_accuracy: 0.6502 - lr: 0.0010
Epoch 17/100
600/600 [==============================] - 6s 10ms/step - loss: 1.1793 -
accuracy: 0.6156 - val_loss: 1.0349 - val_accuracy: 0.6533 - lr: 0.0010
Epoch 18/100
600/600 [==============================] - 5s 9ms/step - loss: 1.1816 -
accuracy: 0.6161 - val_loss: 1.0514 - val_accuracy: 0.6515 - lr: 0.0010
Epoch 19/100
600/600 [==============================] - 6s 10ms/step - loss: 1.1675 -
accuracy: 0.6189 - val_loss: 1.0218 - val_accuracy: 0.6612 - lr: 0.0010
Epoch 20/100
600/600 [==============================] - 6s 11ms/step - loss: 1.1637 -
accuracy: 0.6239 - val_loss: 1.0145 - val_accuracy: 0.6607 - lr: 0.0010
Epoch 21/100
600/600 [==============================] - 5s 9ms/step - loss: 1.1503 -
accuracy: 0.6270 - val_loss: 1.0257 - val_accuracy: 0.6574 - lr: 0.0010
Epoch 22/100
600/600 [==============================] - 6s 10ms/step - loss: 1.1422 -
```

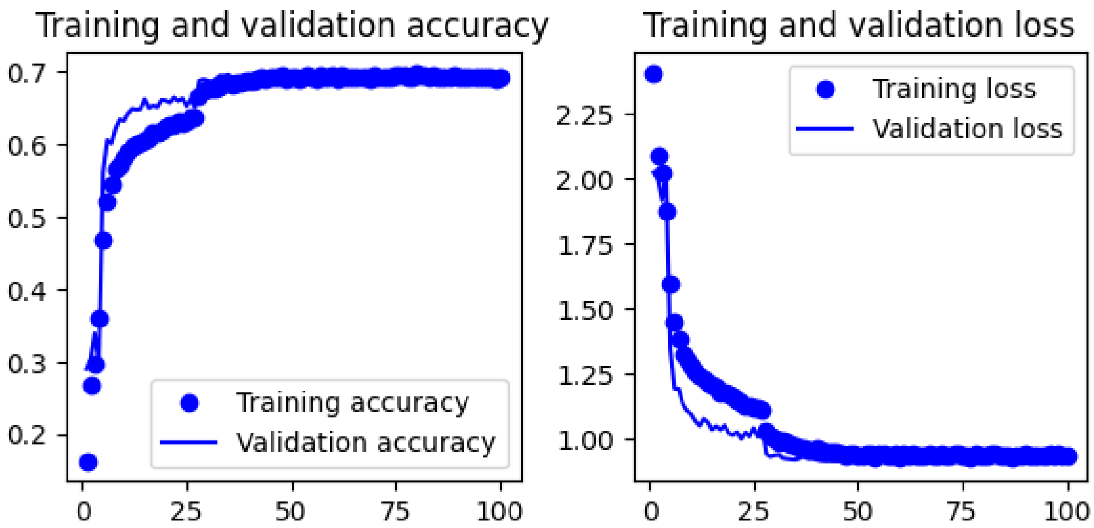
```
accuracy: 0.6275 - val_loss: 1.0006 - val_accuracy: 0.6650 - lr: 0.0010
Epoch 23/100
600/600 [==============================] - 5s 8ms/step - loss: 1.1291 -
accuracy: 0.6318 - val_loss: 1.0249 - val_accuracy: 0.6599 - lr: 0.0010
Epoch 24/100
600/600 [==============================] - 6s 10ms/step - loss: 1.1245 -
accuracy: 0.6307 - val_loss: 1.0094 - val_accuracy: 0.6634 - lr: 0.0010
Epoch 25/100
600/600 [==============================] - 6s 9ms/step - loss: 1.1226 -
accuracy: 0.6332 - val_loss: 1.0411 - val_accuracy: 0.6525 - lr: 0.0010
Epoch 26/100
600/600 [==============================] - 5s 9ms/step - loss: 1.1156 -
accuracy: 0.6389 - val_loss: 1.0078 - val_accuracy: 0.6620 - lr: 0.0010
Epoch 27/100
600/600 [==============================] - 6s 10ms/step - loss: 1.1095 -
accuracy: 0.6388 - val_loss: 1.0511 - val_accuracy: 0.6511 - lr: 0.0010
Epoch 28/100
600/600 [==============================] - 9s 14ms/step - loss: 1.0304 -
accuracy: 0.6657 - val_loss: 0.9416 - val_accuracy: 0.6876 - lr: 1.0000e-04
Epoch 29/100
600/600 [==============================] - 8s 14ms/step - loss: 1.0098 -
accuracy: 0.6738 - val_loss: 0.9319 - val_accuracy: 0.6896 - lr: 1.0000e-04
Epoch 30/100
600/600 [==============================] - 6s 9ms/step - loss: 1.0039 -
accuracy: 0.6759 - val_loss: 0.9366 - val_accuracy: 0.6883 - lr: 1.0000e-04
Epoch 31/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9898 -
accuracy: 0.6767 - val_loss: 0.9393 - val_accuracy: 0.6859 - lr: 1.0000e-04
Epoch 32/100
600/600 [==============================] - 8s 14ms/step - loss: 0.9933 -
accuracy: 0.6780 - val_loss: 0.9258 - val_accuracy: 0.6902 - lr: 1.0000e-04
Epoch 33/100
600/600 [==============================] - 8s 14ms/step - loss: 0.9845 -
accuracy: 0.6793 - val_loss: 0.9232 - val_accuracy: 0.6947 - lr: 1.0000e-04
Epoch 34/100
600/600 [==============================] - 9s 14ms/step - loss: 0.9765 -
accuracy: 0.6859 - val_loss: 0.9212 - val_accuracy: 0.6942 - lr: 1.0000e-04
Epoch 35/100
600/600 [==============================] - 8s 13ms/step - loss: 0.9749 -
accuracy: 0.6849 - val_loss: 0.9200 - val_accuracy: 0.6959 - lr: 1.0000e-04
Epoch 36/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9679 -
accuracy: 0.6827 - val_loss: 0.9252 - val_accuracy: 0.6907 - lr: 1.0000e-04
Epoch 37/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9669 -
accuracy: 0.6843 - val_loss: 0.9393 - val_accuracy: 0.6859 - lr: 1.0000e-04
Epoch 38/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9606 -
```

```
accuracy: 0.6840 - val_loss: 0.9249 - val_accuracy: 0.6927 - lr: 1.0000e-04
Epoch 39/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9573 -
accuracy: 0.6879 - val_loss: 0.9258 - val_accuracy: 0.6910 - lr: 1.0000e-04
Epoch 40/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9621 -
accuracy: 0.6864 - val_loss: 0.9205 - val_accuracy: 0.6931 - lr: 1.0000e-04
Epoch 41/100
600/600 [==============================] - 9s 14ms/step - loss: 0.9572 -
accuracy: 0.6881 - val_loss: 0.9164 - val_accuracy: 0.6943 - lr: 1.0000e-05
Epoch 42/100
600/600 [==============================] - 9s 14ms/step - loss: 0.9472 -
accuracy: 0.6908 - val_loss: 0.9151 - val_accuracy: 0.6939 - lr: 1.0000e-05
Epoch 43/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9468 -
accuracy: 0.6928 - val_loss: 0.9159 - val_accuracy: 0.6936 - lr: 1.0000e-05
Epoch 44/100
600/600 [==============================] - 9s 15ms/step - loss: 0.9505 -
accuracy: 0.6901 - val_loss: 0.9143 - val_accuracy: 0.6934 - lr: 1.0000e-05
Epoch 45/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9486 -
accuracy: 0.6917 - val_loss: 0.9144 - val_accuracy: 0.6935 - lr: 1.0000e-05
Epoch 46/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9414 -
accuracy: 0.6925 - val_loss: 0.9153 - val_accuracy: 0.6943 - lr: 1.0000e-05
Epoch 47/100
600/600 [==============================] - 8s 14ms/step - loss: 0.9372 -
accuracy: 0.6930 - val_loss: 0.9136 - val_accuracy: 0.6944 - lr: 1.0000e-05
Epoch 48/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9403 -
accuracy: 0.6955 - val_loss: 0.9148 - val_accuracy: 0.6954 - lr: 1.0000e-05
Epoch 49/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9435 -
accuracy: 0.6913 - val_loss: 0.9144 - val_accuracy: 0.6956 - lr: 1.0000e-05
Epoch 50/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9355 -
accuracy: 0.6927 - val_loss: 0.9145 - val_accuracy: 0.6950 - lr: 1.0000e-05
Epoch 51/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9381 -
accuracy: 0.6923 - val_loss: 0.9144 - val_accuracy: 0.6938 - lr: 1.0000e-05
Epoch 52/100
600/600 [==============================] - 8s 13ms/step - loss: 0.9436 -
accuracy: 0.6913 - val_loss: 0.9127 - val_accuracy: 0.6952 - lr: 1.0000e-05
Epoch 53/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9409 -
accuracy: 0.6927 - val_loss: 0.9151 - val_accuracy: 0.6938 - lr: 1.0000e-05
Epoch 54/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9316 -
```

```
accuracy: 0.6949 - val_loss: 0.9151 - val_accuracy: 0.6948 - lr: 1.0000e-05
Epoch 55/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9417 -
accuracy: 0.6930 - val_loss: 0.9153 - val_accuracy: 0.6940 - lr: 1.0000e-05
Epoch 56/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9454 -
accuracy: 0.6909 - val_loss: 0.9156 - val_accuracy: 0.6929 - lr: 1.0000e-05
Epoch 57/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9348 -
accuracy: 0.6953 - val_loss: 0.9143 - val_accuracy: 0.6944 - lr: 1.0000e-05
Epoch 58/100
600/600 [==============================] - 7s 12ms/step - loss: 0.9353 -
accuracy: 0.6942 - val_loss: 0.9138 - val_accuracy: 0.6931 - lr: 1.0000e-06
Epoch 59/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9416 -
accuracy: 0.6931 - val_loss: 0.9137 - val_accuracy: 0.6948 - lr: 1.0000e-06
Epoch 60/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9305 -
accuracy: 0.6962 - val_loss: 0.9146 - val_accuracy: 0.6934 - lr: 1.0000e-06
Epoch 61/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9450 -
accuracy: 0.6904 - val_loss: 0.9143 - val_accuracy: 0.6940 - lr: 1.0000e-06
Epoch 62/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9339 -
accuracy: 0.6943 - val_loss: 0.9146 - val_accuracy: 0.6940 - lr: 1.0000e-06
Epoch 63/100
600/600 [==============================] - 6s 11ms/step - loss: 0.9410 -
accuracy: 0.6942 - val_loss: 0.9145 - val_accuracy: 0.6940 - lr: 1.0000e-07
Epoch 64/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9344 -
accuracy: 0.6939 - val_loss: 0.9144 - val_accuracy: 0.6935 - lr: 1.0000e-07
Epoch 65/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9366 -
accuracy: 0.6920 - val_loss: 0.9144 - val_accuracy: 0.6935 - lr: 1.0000e-07
Epoch 66/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9383 -
accuracy: 0.6932 - val_loss: 0.9139 - val_accuracy: 0.6930 - lr: 1.0000e-07
Epoch 67/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9410 -
accuracy: 0.6924 - val_loss: 0.9145 - val_accuracy: 0.6931 - lr: 1.0000e-07
Epoch 68/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9345 -
accuracy: 0.6919 - val_loss: 0.9143 - val_accuracy: 0.6940 - lr: 1.0000e-07
Epoch 69/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9374 -
accuracy: 0.6915 - val_loss: 0.9148 - val_accuracy: 0.6939 - lr: 1.0000e-07
Epoch 70/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9393 -
```

```
accuracy: 0.6954 - val_loss: 0.9140 - val_accuracy: 0.6938 - lr: 1.0000e-07
Epoch 71/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9389 -
accuracy: 0.6932 - val_loss: 0.9141 - val_accuracy: 0.6935 - lr: 1.0000e-07
Epoch 72/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9380 -
accuracy: 0.6936 - val_loss: 0.9138 - val_accuracy: 0.6942 - lr: 1.0000e-07
Epoch 73/100
600/600 [==============================] - 6s 11ms/step - loss: 0.9400 -
accuracy: 0.6917 - val_loss: 0.9142 - val_accuracy: 0.6946 - lr: 1.0000e-07
Epoch 74/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9431 -
accuracy: 0.6917 - val_loss: 0.9142 - val_accuracy: 0.6940 - lr: 1.0000e-07
Epoch 75/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9375 -
accuracy: 0.6943 - val_loss: 0.9140 - val_accuracy: 0.6940 - lr: 1.0000e-07
Epoch 76/100
600/600 [==============================] - 7s 12ms/step - loss: 0.9390 -
accuracy: 0.6947 - val_loss: 0.9143 - val_accuracy: 0.6930 - lr: 1.0000e-07
Epoch 77/100
600/600 [==============================] - 9s 15ms/step - loss: 0.9303 -
accuracy: 0.6943 - val_loss: 0.9139 - val_accuracy: 0.6931 - lr: 1.0000e-07
Epoch 78/100
600/600 [==============================] - 6s 11ms/step - loss: 0.9393 -
accuracy: 0.6925 - val_loss: 0.9140 - val_accuracy: 0.6935 - lr: 1.0000e-07
Epoch 79/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9319 -
accuracy: 0.6938 - val_loss: 0.9145 - val_accuracy: 0.6928 - lr: 1.0000e-07
Epoch 80/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9357 -
accuracy: 0.6995 - val_loss: 0.9145 - val_accuracy: 0.6925 - lr: 1.0000e-07
Epoch 81/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9358 -
accuracy: 0.6955 - val_loss: 0.9141 - val_accuracy: 0.6950 - lr: 1.0000e-07
Epoch 82/100
600/600 [==============================] - 6s 11ms/step - loss: 0.9425 -
accuracy: 0.6956 - val_loss: 0.9143 - val_accuracy: 0.6940 - lr: 1.0000e-07
Epoch 83/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9425 -
accuracy: 0.6929 - val_loss: 0.9144 - val_accuracy: 0.6933 - lr: 1.0000e-07
Epoch 84/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9356 -
accuracy: 0.6961 - val_loss: 0.9144 - val_accuracy: 0.6947 - lr: 1.0000e-07
Epoch 85/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9377 -
accuracy: 0.6938 - val_loss: 0.9138 - val_accuracy: 0.6947 - lr: 1.0000e-07
Epoch 86/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9371 -
```

```
accuracy: 0.6917 - val_loss: 0.9144 - val_accuracy: 0.6939 - lr: 1.0000e-07
Epoch 87/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9312 -
accuracy: 0.6936 - val_loss: 0.9144 - val_accuracy: 0.6940 - lr: 1.0000e-07
Epoch 88/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9348 -
accuracy: 0.6921 - val_loss: 0.9139 - val_accuracy: 0.6935 - lr: 1.0000e-07
Epoch 89/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9347 -
accuracy: 0.6960 - val_loss: 0.9144 - val_accuracy: 0.6940 - lr: 1.0000e-07
Epoch 90/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9396 -
accuracy: 0.6924 - val_loss: 0.9148 - val_accuracy: 0.6939 - lr: 1.0000e-07
Epoch 91/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9357 -
accuracy: 0.6934 - val_loss: 0.9148 - val_accuracy: 0.6932 - lr: 1.0000e-07
Epoch 92/100
600/600 [==============================] - 6s 11ms/step - loss: 0.9380 -
accuracy: 0.6941 - val_loss: 0.9138 - val_accuracy: 0.6950 - lr: 1.0000e-07
Epoch 93/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9372 -
accuracy: 0.6927 - val_loss: 0.9142 - val_accuracy: 0.6939 - lr: 1.0000e-07
Epoch 94/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9363 -
accuracy: 0.6936 - val_loss: 0.9144 - val_accuracy: 0.6943 - lr: 1.0000e-07
Epoch 95/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9325 -
accuracy: 0.6923 - val_loss: 0.9147 - val_accuracy: 0.6941 - lr: 1.0000e-07
Epoch 96/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9404 -
accuracy: 0.6924 - val_loss: 0.9141 - val_accuracy: 0.6938 - lr: 1.0000e-07
Epoch 97/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9364 -
accuracy: 0.6941 - val_loss: 0.9144 - val_accuracy: 0.6934 - lr: 1.0000e-07
Epoch 98/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9399 -
accuracy: 0.6935 - val_loss: 0.9140 - val_accuracy: 0.6933 - lr: 1.0000e-07
Epoch 99/100
600/600 [==============================] - 7s 11ms/step - loss: 0.9370 -
accuracy: 0.6907 - val_loss: 0.9138 - val_accuracy: 0.6941 - lr: 1.0000e-07
Epoch 100/100
600/600 [==============================] - 9s 15ms/step - loss: 0.9385 -
accuracy: 0.6934 - val_loss: 0.9143 - val_accuracy: 0.6942 - lr: 1.0000e-07
```




```
375/375 [==============================] - 4s 8ms/step - loss: 0.9381 -
accuracy: 0.6927
Test accuracy: 0.693
```

Removing layers and changing regularization Gives 69.3% accuracy

```
[ ]: new_inputs = keras.Input(shape=(32, 32, 1))
x = layers.Conv2D(filters = 32, kernel_size = 5, strides = 1, activation =␣
 ↪'relu', input_shape = (32,32,1), kernel_regularizer=l1_l2(0.0005, 0.
 ↪0005))(new_inputs)
x = layers.Conv2D(filters = 32, kernel_size = 5, strides = 1, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size = 2, strides = 2)(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters = 64, kernel_size = 3, strides = 1, activation =␣
 ↪'relu', kernel_regularizer=l1_l2(0.0005, 0.0005))(x)
x = layers.Conv2D(filters = 64, kernel_size = 3, strides = 1, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size = 2, strides = 2)(x)
x = layers.Dropout(0.5)(x)
x = layers.Flatten()(x)
x = layers.Dense(units = 256, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dropout(0.5)(x)
```

```python
# define output layer
new_outputs = layers.Dense(10, activation="softmax")(x)
# define model
model1 = keras.Model(inputs=new_inputs, outputs=new_outputs)
# compile model
model1.compile(optimizer = optimizers.Adam(),
  loss="categorical_crossentropy",
  metrics=["accuracy"])
# train model using validation split of 20% for 100 epochs
history1 = model1.fit(new_X_train,
                    new_y_train,
                    epochs=100,
                    validation_split = 0.2,
                    # callbacks=[early_stopping, variable_learning_rate,␣
 ↪model_checkpoint],
                    callbacks=[variable_learning_rate, model_checkpoint],
                    batch_size=64,
                    verbose=True)
plot_and_print(history1, model1, new_X_test, new_y_test)
```

```
Epoch 1/100
600/600 [==============================] - 12s 11ms/step - loss: 2.6255 -
accuracy: 0.2255 - val_loss: 2.2121 - val_accuracy: 0.3014 - lr: 0.0010
Epoch 2/100
600/600 [==============================] - 7s 12ms/step - loss: 2.1718 -
accuracy: 0.2930 - val_loss: 2.0206 - val_accuracy: 0.3279 - lr: 0.0010
Epoch 3/100
600/600 [==============================] - 9s 14ms/step - loss: 1.8952 -
accuracy: 0.3673 - val_loss: 1.7965 - val_accuracy: 0.4709 - lr: 0.0010
Epoch 4/100
600/600 [==============================] - 6s 11ms/step - loss: 1.5596 -
accuracy: 0.4855 - val_loss: 1.3200 - val_accuracy: 0.5809 - lr: 0.0010
Epoch 5/100
600/600 [==============================] - 6s 11ms/step - loss: 1.4325 -
accuracy: 0.5296 - val_loss: 1.2876 - val_accuracy: 0.5769 - lr: 0.0010
Epoch 6/100
600/600 [==============================] - 6s 10ms/step - loss: 1.3807 -
accuracy: 0.5471 - val_loss: 1.1891 - val_accuracy: 0.6144 - lr: 0.0010
Epoch 7/100
600/600 [==============================] - 4s 7ms/step - loss: 1.3372 -
accuracy: 0.5589 - val_loss: 1.1623 - val_accuracy: 0.6168 - lr: 0.0010
Epoch 8/100
600/600 [==============================] - 5s 9ms/step - loss: 1.3156 -
accuracy: 0.5666 - val_loss: 1.2099 - val_accuracy: 0.6091 - lr: 0.0010
Epoch 9/100
600/600 [==============================] - 5s 8ms/step - loss: 1.3013 -
accuracy: 0.5721 - val_loss: 1.1506 - val_accuracy: 0.6271 - lr: 0.0010
```

```
Epoch 10/100
600/600 [==============================] - 4s 7ms/step - loss: 1.2821 -
accuracy: 0.5784 - val_loss: 1.1413 - val_accuracy: 0.6249 - lr: 0.0010
Epoch 11/100
600/600 [==============================] - 6s 10ms/step - loss: 1.2750 -
accuracy: 0.5814 - val_loss: 1.1166 - val_accuracy: 0.6378 - lr: 0.0010
Epoch 12/100
600/600 [==============================] - 5s 8ms/step - loss: 1.2578 -
accuracy: 0.5866 - val_loss: 1.1266 - val_accuracy: 0.6354 - lr: 0.0010
Epoch 13/100
600/600 [==============================] - 5s 8ms/step - loss: 1.2552 -
accuracy: 0.5863 - val_loss: 1.1418 - val_accuracy: 0.6313 - lr: 0.0010
Epoch 14/100
600/600 [==============================] - 5s 9ms/step - loss: 1.2478 -
accuracy: 0.5907 - val_loss: 1.1228 - val_accuracy: 0.6334 - lr: 0.0010
Epoch 15/100
600/600 [==============================] - 5s 8ms/step - loss: 1.2295 -
accuracy: 0.5952 - val_loss: 1.1048 - val_accuracy: 0.6386 - lr: 0.0010
Epoch 16/100
600/600 [==============================] - 5s 8ms/step - loss: 1.2295 -
accuracy: 0.5968 - val_loss: 1.1104 - val_accuracy: 0.6390 - lr: 0.0010
Epoch 17/100
600/600 [==============================] - 5s 9ms/step - loss: 1.2213 -
accuracy: 0.5980 - val_loss: 1.1764 - val_accuracy: 0.6082 - lr: 0.0010
Epoch 18/100
600/600 [==============================] - 5s 8ms/step - loss: 1.2166 -
accuracy: 0.5985 - val_loss: 1.0937 - val_accuracy: 0.6423 - lr: 0.0010
Epoch 19/100
600/600 [==============================] - 5s 9ms/step - loss: 1.2079 -
accuracy: 0.6018 - val_loss: 1.0853 - val_accuracy: 0.6421 - lr: 0.0010
Epoch 20/100
600/600 [==============================] - 5s 8ms/step - loss: 1.2041 -
accuracy: 0.6026 - val_loss: 1.0984 - val_accuracy: 0.6428 - lr: 0.0010
Epoch 21/100
600/600 [==============================] - 5s 8ms/step - loss: 1.2024 -
accuracy: 0.6047 - val_loss: 1.1215 - val_accuracy: 0.6297 - lr: 0.0010
Epoch 22/100
600/600 [==============================] - 7s 12ms/step - loss: 1.1999 -
accuracy: 0.6073 - val_loss: 1.0915 - val_accuracy: 0.6453 - lr: 0.0010
Epoch 23/100
600/600 [==============================] - 7s 11ms/step - loss: 1.1889 -
accuracy: 0.6073 - val_loss: 1.0726 - val_accuracy: 0.6524 - lr: 0.0010
Epoch 24/100
600/600 [==============================] - 10s 16ms/step - loss: 1.1822 -
accuracy: 0.6091 - val_loss: 1.0488 - val_accuracy: 0.6617 - lr: 0.0010
Epoch 25/100
600/600 [==============================] - 6s 10ms/step - loss: 1.1819 -
accuracy: 0.6108 - val_loss: 1.0599 - val_accuracy: 0.6560 - lr: 0.0010
```

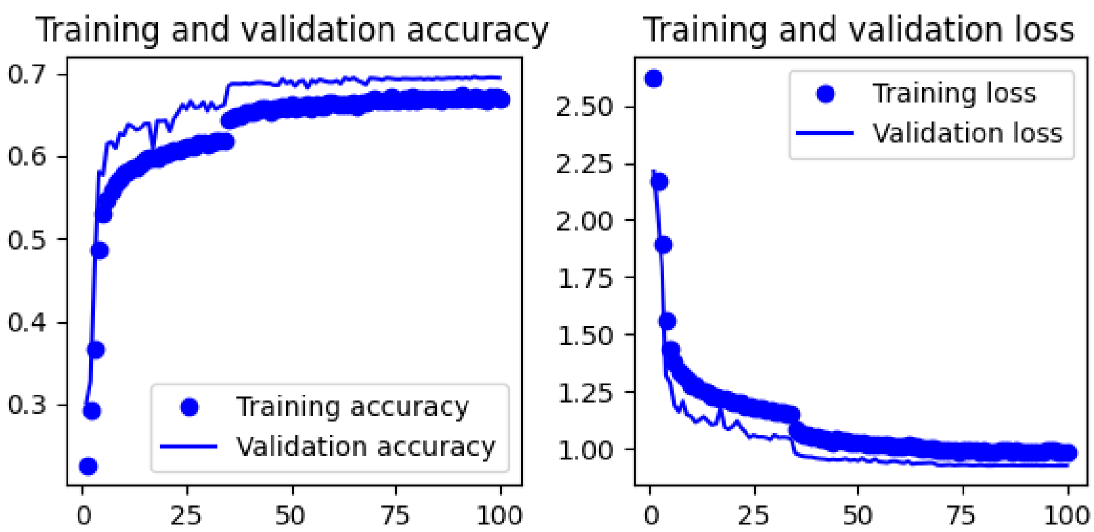
```
Epoch 26/100
600/600 [==============================] - 9s 15ms/step - loss: 1.1794 -
accuracy: 0.6120 - val_loss: 1.0517 - val_accuracy: 0.6657 - lr: 0.0010
Epoch 27/100
600/600 [==============================] - 6s 10ms/step - loss: 1.1758 -
accuracy: 0.6115 - val_loss: 1.0524 - val_accuracy: 0.6579 - lr: 0.0010
Epoch 28/100
600/600 [==============================] - 6s 11ms/step - loss: 1.1683 -
accuracy: 0.6167 - val_loss: 1.0485 - val_accuracy: 0.6586 - lr: 0.0010
Epoch 29/100
600/600 [==============================] - 5s 8ms/step - loss: 1.1662 -
accuracy: 0.6157 - val_loss: 1.0404 - val_accuracy: 0.6647 - lr: 0.0010
Epoch 30/100
600/600 [==============================] - 5s 8ms/step - loss: 1.1641 -
accuracy: 0.6136 - val_loss: 1.0612 - val_accuracy: 0.6540 - lr: 0.0010
Epoch 31/100
600/600 [==============================] - 5s 9ms/step - loss: 1.1639 -
accuracy: 0.6164 - val_loss: 1.0488 - val_accuracy: 0.6641 - lr: 0.0010
Epoch 32/100
600/600 [==============================] - 4s 7ms/step - loss: 1.1600 -
accuracy: 0.6189 - val_loss: 1.0510 - val_accuracy: 0.6565 - lr: 0.0010
Epoch 33/100
600/600 [==============================] - 5s 8ms/step - loss: 1.1544 -
accuracy: 0.6186 - val_loss: 1.0493 - val_accuracy: 0.6593 - lr: 0.0010
Epoch 34/100
600/600 [==============================] - 6s 10ms/step - loss: 1.1562 -
accuracy: 0.6179 - val_loss: 1.0413 - val_accuracy: 0.6616 - lr: 0.0010
Epoch 35/100
600/600 [==============================] - 5s 8ms/step - loss: 1.0880 -
accuracy: 0.6430 - val_loss: 0.9814 - val_accuracy: 0.6846 - lr: 1.0000e-04
Epoch 36/100
600/600 [==============================] - 5s 9ms/step - loss: 1.0675 -
accuracy: 0.6467 - val_loss: 0.9721 - val_accuracy: 0.6871 - lr: 1.0000e-04
Epoch 37/100
600/600 [==============================] - 5s 8ms/step - loss: 1.0624 -
accuracy: 0.6490 - val_loss: 0.9669 - val_accuracy: 0.6870 - lr: 1.0000e-04
Epoch 38/100
600/600 [==============================] - 5s 8ms/step - loss: 1.0604 -
accuracy: 0.6490 - val_loss: 0.9645 - val_accuracy: 0.6869 - lr: 1.0000e-04
Epoch 39/100
600/600 [==============================] - 6s 9ms/step - loss: 1.0505 -
accuracy: 0.6531 - val_loss: 0.9637 - val_accuracy: 0.6876 - lr: 1.0000e-04
Epoch 40/100
600/600 [==============================] - 4s 7ms/step - loss: 1.0510 -
accuracy: 0.6523 - val_loss: 0.9591 - val_accuracy: 0.6869 - lr: 1.0000e-04
Epoch 41/100
600/600 [==============================] - 5s 8ms/step - loss: 1.0472 -
accuracy: 0.6530 - val_loss: 0.9566 - val_accuracy: 0.6876 - lr: 1.0000e-04
```

```
Epoch 42/100
600/600 [==============================] - 6s 9ms/step - loss: 1.0419 -
accuracy: 0.6561 - val_loss: 0.9554 - val_accuracy: 0.6886 - lr: 1.0000e-04
Epoch 43/100
600/600 [==============================] - 5s 8ms/step - loss: 1.0383 -
accuracy: 0.6574 - val_loss: 0.9511 - val_accuracy: 0.6884 - lr: 1.0000e-04
Epoch 44/100
600/600 [==============================] - 5s 8ms/step - loss: 1.0314 -
accuracy: 0.6578 - val_loss: 0.9546 - val_accuracy: 0.6875 - lr: 1.0000e-04
Epoch 45/100
600/600 [==============================] - 5s 9ms/step - loss: 1.0416 -
accuracy: 0.6535 - val_loss: 0.9518 - val_accuracy: 0.6873 - lr: 1.0000e-04
Epoch 46/100
600/600 [==============================] - 5s 8ms/step - loss: 1.0327 -
accuracy: 0.6583 - val_loss: 0.9528 - val_accuracy: 0.6855 - lr: 1.0000e-04
Epoch 47/100
600/600 [==============================] - 5s 8ms/step - loss: 1.0326 -
accuracy: 0.6565 - val_loss: 0.9579 - val_accuracy: 0.6854 - lr: 1.0000e-04
Epoch 48/100
600/600 [==============================] - 5s 8ms/step - loss: 1.0287 -
accuracy: 0.6605 - val_loss: 0.9494 - val_accuracy: 0.6928 - lr: 1.0000e-04
Epoch 49/100
600/600 [==============================] - 5s 8ms/step - loss: 1.0317 -
accuracy: 0.6576 - val_loss: 0.9531 - val_accuracy: 0.6878 - lr: 1.0000e-04
Epoch 50/100
600/600 [==============================] - 8s 13ms/step - loss: 1.0258 -
accuracy: 0.6618 - val_loss: 0.9477 - val_accuracy: 0.6901 - lr: 1.0000e-04
Epoch 51/100
600/600 [==============================] - 5s 9ms/step - loss: 1.0291 -
accuracy: 0.6583 - val_loss: 0.9589 - val_accuracy: 0.6840 - lr: 1.0000e-04
Epoch 52/100
600/600 [==============================] - 8s 13ms/step - loss: 1.0188 -
accuracy: 0.6606 - val_loss: 0.9470 - val_accuracy: 0.6897 - lr: 1.0000e-04
Epoch 53/100
600/600 [==============================] - 9s 14ms/step - loss: 1.0210 -
accuracy: 0.6604 - val_loss: 0.9446 - val_accuracy: 0.6909 - lr: 1.0000e-04
Epoch 54/100
600/600 [==============================] - 8s 13ms/step - loss: 1.0184 -
accuracy: 0.6615 - val_loss: 0.9567 - val_accuracy: 0.6821 - lr: 1.0000e-04
Epoch 55/100
600/600 [==============================] - 6s 11ms/step - loss: 1.0226 -
accuracy: 0.6576 - val_loss: 0.9441 - val_accuracy: 0.6913 - lr: 1.0000e-04
Epoch 56/100
600/600 [==============================] - 7s 12ms/step - loss: 1.0173 -
accuracy: 0.6623 - val_loss: 0.9421 - val_accuracy: 0.6883 - lr: 1.0000e-04
Epoch 57/100
600/600 [==============================] - 6s 9ms/step - loss: 1.0174 -
accuracy: 0.6595 - val_loss: 0.9401 - val_accuracy: 0.6922 - lr: 1.0000e-04
```

```
Epoch 58/100
600/600 [==============================] - 5s 8ms/step - loss: 1.0137 -
accuracy: 0.6588 - val_loss: 0.9459 - val_accuracy: 0.6903 - lr: 1.0000e-04
Epoch 59/100
600/600 [==============================] - 5s 9ms/step - loss: 1.0116 -
accuracy: 0.6634 - val_loss: 0.9460 - val_accuracy: 0.6892 - lr: 1.0000e-04
Epoch 60/100
600/600 [==============================] - 5s 8ms/step - loss: 1.0144 -
accuracy: 0.6642 - val_loss: 0.9417 - val_accuracy: 0.6890 - lr: 1.0000e-04
Epoch 61/100
600/600 [==============================] - 5s 8ms/step - loss: 1.0116 -
accuracy: 0.6618 - val_loss: 0.9383 - val_accuracy: 0.6907 - lr: 1.0000e-04
Epoch 62/100
600/600 [==============================] - 5s 9ms/step - loss: 1.0170 -
accuracy: 0.6614 - val_loss: 0.9454 - val_accuracy: 0.6873 - lr: 1.0000e-04
Epoch 63/100
600/600 [==============================] - 5s 8ms/step - loss: 1.0107 -
accuracy: 0.6632 - val_loss: 0.9337 - val_accuracy: 0.6942 - lr: 1.0000e-04
Epoch 64/100
600/600 [==============================] - 5s 9ms/step - loss: 1.0134 -
accuracy: 0.6620 - val_loss: 0.9369 - val_accuracy: 0.6918 - lr: 1.0000e-04
Epoch 65/100
600/600 [==============================] - 5s 8ms/step - loss: 1.0053 -
accuracy: 0.6636 - val_loss: 0.9370 - val_accuracy: 0.6941 - lr: 1.0000e-04
Epoch 66/100
600/600 [==============================] - 5s 8ms/step - loss: 1.0057 -
accuracy: 0.6608 - val_loss: 0.9377 - val_accuracy: 0.6907 - lr: 1.0000e-04
Epoch 67/100
600/600 [==============================] - 6s 9ms/step - loss: 1.0039 -
accuracy: 0.6634 - val_loss: 0.9391 - val_accuracy: 0.6892 - lr: 1.0000e-04
Epoch 68/100
600/600 [==============================] - 5s 7ms/step - loss: 1.0065 -
accuracy: 0.6642 - val_loss: 0.9386 - val_accuracy: 0.6860 - lr: 1.0000e-04
Epoch 69/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9973 -
accuracy: 0.6662 - val_loss: 0.9294 - val_accuracy: 0.6938 - lr: 1.0000e-05
Epoch 70/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9967 -
accuracy: 0.6679 - val_loss: 0.9282 - val_accuracy: 0.6933 - lr: 1.0000e-05
Epoch 71/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9926 -
accuracy: 0.6672 - val_loss: 0.9295 - val_accuracy: 0.6926 - lr: 1.0000e-05
Epoch 72/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9974 -
accuracy: 0.6660 - val_loss: 0.9308 - val_accuracy: 0.6916 - lr: 1.0000e-05
Epoch 73/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9931 -
accuracy: 0.6677 - val_loss: 0.9280 - val_accuracy: 0.6940 - lr: 1.0000e-05
```

```
Epoch 74/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9873 -
accuracy: 0.6712 - val_loss: 0.9292 - val_accuracy: 0.6933 - lr: 1.0000e-05
Epoch 75/100
600/600 [==============================] - 8s 14ms/step - loss: 0.9905 -
accuracy: 0.6670 - val_loss: 0.9299 - val_accuracy: 0.6925 - lr: 1.0000e-05
Epoch 76/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9940 -
accuracy: 0.6676 - val_loss: 0.9300 - val_accuracy: 0.6923 - lr: 1.0000e-05
Epoch 77/100
600/600 [==============================] - 8s 13ms/step - loss: 0.9935 -
accuracy: 0.6686 - val_loss: 0.9280 - val_accuracy: 0.6931 - lr: 1.0000e-05
Epoch 78/100
600/600 [==============================] - 10s 16ms/step - loss: 0.9925 -
accuracy: 0.6666 - val_loss: 0.9277 - val_accuracy: 0.6930 - lr: 1.0000e-05
Epoch 79/100
600/600 [==============================] - 9s 15ms/step - loss: 0.9845 -
accuracy: 0.6717 - val_loss: 0.9292 - val_accuracy: 0.6927 - lr: 1.0000e-05
Epoch 80/100
600/600 [==============================] - 7s 11ms/step - loss: 0.9909 -
accuracy: 0.6666 - val_loss: 0.9286 - val_accuracy: 0.6928 - lr: 1.0000e-05
Epoch 81/100
600/600 [==============================] - 7s 12ms/step - loss: 0.9899 -
accuracy: 0.6695 - val_loss: 0.9269 - val_accuracy: 0.6932 - lr: 1.0000e-05
Epoch 82/100
600/600 [==============================] - 7s 11ms/step - loss: 0.9878 -
accuracy: 0.6679 - val_loss: 0.9269 - val_accuracy: 0.6923 - lr: 1.0000e-05
Epoch 83/100
600/600 [==============================] - 7s 11ms/step - loss: 0.9891 -
accuracy: 0.6680 - val_loss: 0.9282 - val_accuracy: 0.6930 - lr: 1.0000e-05
Epoch 84/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9927 -
accuracy: 0.6679 - val_loss: 0.9282 - val_accuracy: 0.6922 - lr: 1.0000e-05
Epoch 85/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9911 -
accuracy: 0.6685 - val_loss: 0.9281 - val_accuracy: 0.6938 - lr: 1.0000e-05
Epoch 86/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9913 -
accuracy: 0.6672 - val_loss: 0.9283 - val_accuracy: 0.6928 - lr: 1.0000e-05
Epoch 87/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9846 -
accuracy: 0.6687 - val_loss: 0.9282 - val_accuracy: 0.6933 - lr: 1.0000e-06
Epoch 88/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9902 -
accuracy: 0.6695 - val_loss: 0.9279 - val_accuracy: 0.6939 - lr: 1.0000e-06
Epoch 89/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9896 -
accuracy: 0.6701 - val_loss: 0.9276 - val_accuracy: 0.6936 - lr: 1.0000e-06
```

```
Epoch 90/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9892 -
accuracy: 0.6679 - val_loss: 0.9273 - val_accuracy: 0.6944 - lr: 1.0000e-06
Epoch 91/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9830 -
accuracy: 0.6730 - val_loss: 0.9275 - val_accuracy: 0.6933 - lr: 1.0000e-06
Epoch 92/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9879 -
accuracy: 0.6695 - val_loss: 0.9273 - val_accuracy: 0.6944 - lr: 1.0000e-07
Epoch 93/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9883 -
accuracy: 0.6697 - val_loss: 0.9277 - val_accuracy: 0.6933 - lr: 1.0000e-07
Epoch 94/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9888 -
accuracy: 0.6703 - val_loss: 0.9267 - val_accuracy: 0.6955 - lr: 1.0000e-07
Epoch 95/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9902 -
accuracy: 0.6696 - val_loss: 0.9273 - val_accuracy: 0.6940 - lr: 1.0000e-07
Epoch 96/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9933 -
accuracy: 0.6681 - val_loss: 0.9274 - val_accuracy: 0.6938 - lr: 1.0000e-07
Epoch 97/100
600/600 [==============================] - 8s 13ms/step - loss: 0.9931 -
accuracy: 0.6663 - val_loss: 0.9270 - val_accuracy: 0.6944 - lr: 1.0000e-07
Epoch 98/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9889 -
accuracy: 0.6713 - val_loss: 0.9270 - val_accuracy: 0.6946 - lr: 1.0000e-07
Epoch 99/100
600/600 [==============================] - 7s 11ms/step - loss: 0.9878 -
accuracy: 0.6704 - val_loss: 0.9270 - val_accuracy: 0.6944 - lr: 1.0000e-07
Epoch 100/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9863 -
accuracy: 0.6681 - val_loss: 0.9272 - val_accuracy: 0.6941 - lr: 1.0000e-07
```




```
375/375 [==============================] - 2s 4ms/step - loss: 0.9348 -
accuracy: 0.6928
Test accuracy: 0.693
```

2.4 Set 4: Inspiration from this article.

Add Gaussian Noise as layers. Gives 69.3% accuracy

[ ]:
```
new_inputs = keras.Input(shape=(32, 32, 1))
x = layers.Conv2D(filters = 32, kernel_size = 5, strides = 1, activation =␣
↪'relu', input_shape = (32,32,1), kernel_regularizer=l2(0.0005))(new_inputs)
x = layers.Conv2D(filters = 32, kernel_size = 5, strides = 1, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size = 2, strides = 2)(x)
x = layers.GaussianNoise(0.1)(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters = 64, kernel_size = 3, strides = 1, activation =␣
↪'relu', kernel_regularizer=l2(0.0005))(x)
x = layers.Conv2D(filters = 64, kernel_size = 3, strides = 1, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size = 2, strides = 2)(x)
x = layers.GaussianNoise(0.1)(x)
x = layers.Dropout(0.5)(x)
x = layers.Flatten()(x)
x = layers.Dense(units = 256, use_bias=False)(x)
x = layers.BatchNormalization()(x)
```

```
x = layers.Activation('relu')(x)
x = layers.Dense(units = 128, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dense(units = 64, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dropout(0.5)(x)

# define output layer
new_outputs = layers.Dense(10, activation="softmax")(x)
# define model
model1 = keras.Model(inputs=new_inputs, outputs=new_outputs)
# compile model
model1.compile(optimizer = optimizers.Adam(),
  loss="categorical_crossentropy",
  metrics=["accuracy"])
# train model using validation split of 20% for 100 epochs
history1 = model1.fit(new_X_train,
                    new_y_train,
                    epochs=100,
                    validation_split = 0.2,
                    # callbacks=[early_stopping, variable_learning_rate,␣
 ↪model_checkpoint],
                    callbacks=[variable_learning_rate, model_checkpoint],
                    batch_size=64,
                    verbose=True)
plot_and_print(history1, model1, new_X_test, new_y_test)
```

```
Epoch 1/100
600/600 [==============================] - 18s 14ms/step - loss: 2.3568 -
accuracy: 0.1831 - val_loss: 2.0030 - val_accuracy: 0.2943 - lr: 0.0010
Epoch 2/100
600/600 [==============================] - 10s 16ms/step - loss: 2.0759 -
accuracy: 0.2748 - val_loss: 1.9978 - val_accuracy: 0.2974 - lr: 0.0010
Epoch 3/100
600/600 [==============================] - 9s 15ms/step - loss: 2.0016 -
accuracy: 0.3064 - val_loss: 1.9033 - val_accuracy: 0.3492 - lr: 0.0010
Epoch 4/100
600/600 [==============================] - 8s 13ms/step - loss: 1.7876 -
accuracy: 0.3958 - val_loss: 1.5437 - val_accuracy: 0.5052 - lr: 0.0010
Epoch 5/100
600/600 [==============================] - 8s 14ms/step - loss: 1.5524 -
accuracy: 0.4843 - val_loss: 1.2784 - val_accuracy: 0.5769 - lr: 0.0010
Epoch 6/100
600/600 [==============================] - 7s 11ms/step - loss: 1.4387 -
accuracy: 0.5271 - val_loss: 1.1852 - val_accuracy: 0.6115 - lr: 0.0010
```

```
Epoch 7/100
600/600 [==============================] - 6s 11ms/step - loss: 1.3862 -
accuracy: 0.5455 - val_loss: 1.1160 - val_accuracy: 0.6288 - lr: 0.0010
Epoch 8/100
600/600 [==============================] - 5s 9ms/step - loss: 1.3401 -
accuracy: 0.5630 - val_loss: 1.1584 - val_accuracy: 0.6250 - lr: 0.0010
Epoch 9/100
600/600 [==============================] - 6s 10ms/step - loss: 1.3059 -
accuracy: 0.5728 - val_loss: 1.1561 - val_accuracy: 0.6131 - lr: 0.0010
Epoch 10/100
600/600 [==============================] - 5s 9ms/step - loss: 1.2868 -
accuracy: 0.5797 - val_loss: 1.0755 - val_accuracy: 0.6391 - lr: 0.0010
Epoch 11/100
600/600 [==============================] - 6s 10ms/step - loss: 1.2671 -
accuracy: 0.5871 - val_loss: 1.0579 - val_accuracy: 0.6455 - lr: 0.0010
Epoch 12/100
600/600 [==============================] - 7s 12ms/step - loss: 1.2489 -
accuracy: 0.5921 - val_loss: 1.0492 - val_accuracy: 0.6472 - lr: 0.0010
Epoch 13/100
600/600 [==============================] - 5s 9ms/step - loss: 1.2378 -
accuracy: 0.5973 - val_loss: 1.0602 - val_accuracy: 0.6492 - lr: 0.0010
Epoch 14/100
600/600 [==============================] - 6s 9ms/step - loss: 1.2203 -
accuracy: 0.6026 - val_loss: 1.0387 - val_accuracy: 0.6542 - lr: 0.0010
Epoch 15/100
600/600 [==============================] - 6s 10ms/step - loss: 1.2106 -
accuracy: 0.6055 - val_loss: 1.0305 - val_accuracy: 0.6582 - lr: 0.0010
Epoch 16/100
600/600 [==============================] - 5s 9ms/step - loss: 1.1982 -
accuracy: 0.6073 - val_loss: 1.0441 - val_accuracy: 0.6499 - lr: 0.0010
Epoch 17/100
600/600 [==============================] - 9s 15ms/step - loss: 1.1840 -
accuracy: 0.6172 - val_loss: 1.0461 - val_accuracy: 0.6473 - lr: 0.0010
Epoch 18/100
600/600 [==============================] - 11s 19ms/step - loss: 1.1839 -
accuracy: 0.6131 - val_loss: 1.0539 - val_accuracy: 0.6460 - lr: 0.0010
Epoch 19/100
600/600 [==============================] - 10s 16ms/step - loss: 1.1743 -
accuracy: 0.6145 - val_loss: 1.0331 - val_accuracy: 0.6547 - lr: 0.0010
Epoch 20/100
600/600 [==============================] - 8s 14ms/step - loss: 1.1671 -
accuracy: 0.6187 - val_loss: 1.0181 - val_accuracy: 0.6637 - lr: 0.0010
Epoch 21/100
600/600 [==============================] - 9s 15ms/step - loss: 1.1609 -
accuracy: 0.6210 - val_loss: 1.0331 - val_accuracy: 0.6549 - lr: 0.0010
Epoch 22/100
600/600 [==============================] - 8s 13ms/step - loss: 1.1527 -
accuracy: 0.6255 - val_loss: 1.0178 - val_accuracy: 0.6627 - lr: 0.0010
```

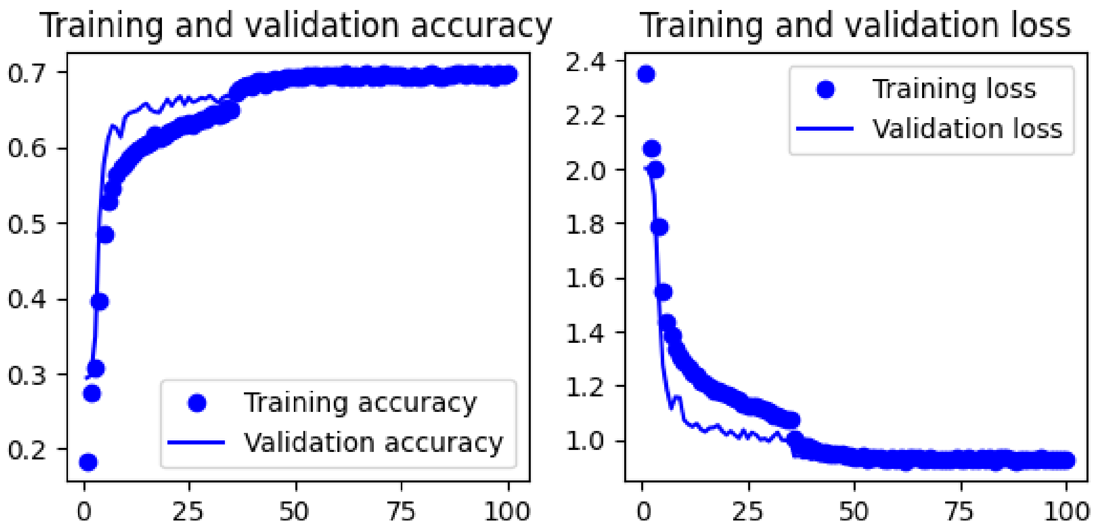
```
Epoch 23/100
600/600 [==============================] - 9s 14ms/step - loss: 1.1444 -
accuracy: 0.6288 - val_loss: 1.0069 - val_accuracy: 0.6674 - lr: 0.0010
Epoch 24/100
600/600 [==============================] - 6s 10ms/step - loss: 1.1317 -
accuracy: 0.6299 - val_loss: 1.0370 - val_accuracy: 0.6562 - lr: 0.0010
Epoch 25/100
600/600 [==============================] - 6s 10ms/step - loss: 1.1286 -
accuracy: 0.6313 - val_loss: 1.0055 - val_accuracy: 0.6662 - lr: 0.0010
Epoch 26/100
600/600 [==============================] - 5s 9ms/step - loss: 1.1287 -
accuracy: 0.6303 - val_loss: 1.0283 - val_accuracy: 0.6596 - lr: 0.0010
Epoch 27/100
600/600 [==============================] - 6s 9ms/step - loss: 1.1244 -
accuracy: 0.6348 - val_loss: 1.0181 - val_accuracy: 0.6622 - lr: 0.0010
Epoch 28/100
600/600 [==============================] - 8s 13ms/step - loss: 1.1179 -
accuracy: 0.6372 - val_loss: 1.0012 - val_accuracy: 0.6657 - lr: 0.0010
Epoch 29/100
600/600 [==============================] - 5s 9ms/step - loss: 1.1103 -
accuracy: 0.6372 - val_loss: 1.0051 - val_accuracy: 0.6643 - lr: 0.0010
Epoch 30/100
600/600 [==============================] - 6s 10ms/step - loss: 1.1022 -
accuracy: 0.6435 - val_loss: 0.9954 - val_accuracy: 0.6682 - lr: 0.0010
Epoch 31/100
600/600 [==============================] - 5s 9ms/step - loss: 1.0927 -
accuracy: 0.6446 - val_loss: 1.0119 - val_accuracy: 0.6635 - lr: 0.0010
Epoch 32/100
600/600 [==============================] - 6s 9ms/step - loss: 1.0920 -
accuracy: 0.6427 - val_loss: 1.0287 - val_accuracy: 0.6586 - lr: 0.0010
Epoch 33/100
600/600 [==============================] - 6s 10ms/step - loss: 1.0846 -
accuracy: 0.6446 - val_loss: 1.0014 - val_accuracy: 0.6657 - lr: 0.0010
Epoch 34/100
600/600 [==============================] - 5s 9ms/step - loss: 1.0769 -
accuracy: 0.6524 - val_loss: 0.9987 - val_accuracy: 0.6685 - lr: 0.0010
Epoch 35/100
600/600 [==============================] - 7s 11ms/step - loss: 1.0785 -
accuracy: 0.6513 - val_loss: 1.0049 - val_accuracy: 0.6666 - lr: 0.0010
Epoch 36/100
600/600 [==============================] - 5s 9ms/step - loss: 1.0023 -
accuracy: 0.6732 - val_loss: 0.9390 - val_accuracy: 0.6832 - lr: 1.0000e-04
Epoch 37/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9835 -
accuracy: 0.6783 - val_loss: 0.9412 - val_accuracy: 0.6873 - lr: 1.0000e-04
Epoch 38/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9753 -
accuracy: 0.6807 - val_loss: 0.9344 - val_accuracy: 0.6898 - lr: 1.0000e-04
```

```
Epoch 39/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9746 -
accuracy: 0.6819 - val_loss: 0.9353 - val_accuracy: 0.6901 - lr: 1.0000e-04
Epoch 40/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9711 -
accuracy: 0.6817 - val_loss: 0.9283 - val_accuracy: 0.6929 - lr: 1.0000e-04
Epoch 41/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9627 -
accuracy: 0.6869 - val_loss: 0.9332 - val_accuracy: 0.6916 - lr: 1.0000e-04
Epoch 42/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9576 -
accuracy: 0.6872 - val_loss: 0.9252 - val_accuracy: 0.6918 - lr: 1.0000e-04
Epoch 43/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9591 -
accuracy: 0.6839 - val_loss: 0.9228 - val_accuracy: 0.6938 - lr: 1.0000e-04
Epoch 44/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9511 -
accuracy: 0.6882 - val_loss: 0.9279 - val_accuracy: 0.6934 - lr: 1.0000e-04
Epoch 45/100
600/600 [==============================] - 7s 12ms/step - loss: 0.9510 -
accuracy: 0.6896 - val_loss: 0.9283 - val_accuracy: 0.6929 - lr: 1.0000e-04
Epoch 46/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9507 -
accuracy: 0.6886 - val_loss: 0.9383 - val_accuracy: 0.6892 - lr: 1.0000e-04
Epoch 47/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9485 -
accuracy: 0.6917 - val_loss: 0.9293 - val_accuracy: 0.6927 - lr: 1.0000e-04
Epoch 48/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9399 -
accuracy: 0.6933 - val_loss: 0.9254 - val_accuracy: 0.6931 - lr: 1.0000e-04
Epoch 49/100
600/600 [==============================] - 7s 11ms/step - loss: 0.9439 -
accuracy: 0.6926 - val_loss: 0.9229 - val_accuracy: 0.6950 - lr: 1.0000e-05
Epoch 50/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9350 -
accuracy: 0.6927 - val_loss: 0.9238 - val_accuracy: 0.6941 - lr: 1.0000e-05
Epoch 51/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9320 -
accuracy: 0.6936 - val_loss: 0.9232 - val_accuracy: 0.6944 - lr: 1.0000e-05
Epoch 52/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9393 -
accuracy: 0.6932 - val_loss: 0.9220 - val_accuracy: 0.6939 - lr: 1.0000e-05
Epoch 53/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9307 -
accuracy: 0.6956 - val_loss: 0.9204 - val_accuracy: 0.6952 - lr: 1.0000e-05
Epoch 54/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9321 -
accuracy: 0.6963 - val_loss: 0.9229 - val_accuracy: 0.6934 - lr: 1.0000e-05
```

```
Epoch 55/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9318 -
accuracy: 0.6962 - val_loss: 0.9253 - val_accuracy: 0.6931 - lr: 1.0000e-05
Epoch 56/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9337 -
accuracy: 0.6939 - val_loss: 0.9252 - val_accuracy: 0.6939 - lr: 1.0000e-05
Epoch 57/100
600/600 [==============================] - 6s 11ms/step - loss: 0.9311 -
accuracy: 0.6959 - val_loss: 0.9236 - val_accuracy: 0.6948 - lr: 1.0000e-05
Epoch 58/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9277 -
accuracy: 0.6955 - val_loss: 0.9238 - val_accuracy: 0.6953 - lr: 1.0000e-05
Epoch 59/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9363 -
accuracy: 0.6952 - val_loss: 0.9234 - val_accuracy: 0.6950 - lr: 1.0000e-06
Epoch 60/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9281 -
accuracy: 0.6946 - val_loss: 0.9232 - val_accuracy: 0.6952 - lr: 1.0000e-06
Epoch 61/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9329 -
accuracy: 0.6934 - val_loss: 0.9221 - val_accuracy: 0.6951 - lr: 1.0000e-06
Epoch 62/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9245 -
accuracy: 0.6979 - val_loss: 0.9235 - val_accuracy: 0.6944 - lr: 1.0000e-06
Epoch 63/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9336 -
accuracy: 0.6943 - val_loss: 0.9220 - val_accuracy: 0.6959 - lr: 1.0000e-06
Epoch 64/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9341 -
accuracy: 0.6960 - val_loss: 0.9231 - val_accuracy: 0.6953 - lr: 1.0000e-07
Epoch 65/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9319 -
accuracy: 0.6943 - val_loss: 0.9226 - val_accuracy: 0.6949 - lr: 1.0000e-07
Epoch 66/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9294 -
accuracy: 0.6969 - val_loss: 0.9226 - val_accuracy: 0.6954 - lr: 1.0000e-07
Epoch 67/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9285 -
accuracy: 0.6979 - val_loss: 0.9223 - val_accuracy: 0.6942 - lr: 1.0000e-07
Epoch 68/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9332 -
accuracy: 0.6941 - val_loss: 0.9230 - val_accuracy: 0.6943 - lr: 1.0000e-07
Epoch 69/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9295 -
accuracy: 0.6965 - val_loss: 0.9219 - val_accuracy: 0.6952 - lr: 1.0000e-07
Epoch 70/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9263 -
accuracy: 0.6935 - val_loss: 0.9232 - val_accuracy: 0.6952 - lr: 1.0000e-07
```

```
Epoch 71/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9269 -
accuracy: 0.6975 - val_loss: 0.9229 - val_accuracy: 0.6964 - lr: 1.0000e-07
Epoch 72/100
600/600 [==============================] - 9s 16ms/step - loss: 0.9283 -
accuracy: 0.6967 - val_loss: 0.9227 - val_accuracy: 0.6959 - lr: 1.0000e-07
Epoch 73/100
600/600 [==============================] - 8s 14ms/step - loss: 0.9278 -
accuracy: 0.6969 - val_loss: 0.9223 - val_accuracy: 0.6953 - lr: 1.0000e-07
Epoch 74/100
600/600 [==============================] - 7s 11ms/step - loss: 0.9326 -
accuracy: 0.6949 - val_loss: 0.9229 - val_accuracy: 0.6947 - lr: 1.0000e-07
Epoch 75/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9299 -
accuracy: 0.6960 - val_loss: 0.9221 - val_accuracy: 0.6948 - lr: 1.0000e-07
Epoch 76/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9288 -
accuracy: 0.6937 - val_loss: 0.9227 - val_accuracy: 0.6950 - lr: 1.0000e-07
Epoch 77/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9343 -
accuracy: 0.6943 - val_loss: 0.9225 - val_accuracy: 0.6957 - lr: 1.0000e-07
Epoch 78/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9277 -
accuracy: 0.6964 - val_loss: 0.9227 - val_accuracy: 0.6954 - lr: 1.0000e-07
Epoch 79/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9281 -
accuracy: 0.6941 - val_loss: 0.9222 - val_accuracy: 0.6948 - lr: 1.0000e-07
Epoch 80/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9364 -
accuracy: 0.6933 - val_loss: 0.9234 - val_accuracy: 0.6945 - lr: 1.0000e-07
Epoch 81/100
600/600 [==============================] - 6s 11ms/step - loss: 0.9303 -
accuracy: 0.6949 - val_loss: 0.9225 - val_accuracy: 0.6940 - lr: 1.0000e-07
Epoch 82/100
600/600 [==============================] - 9s 14ms/step - loss: 0.9339 -
accuracy: 0.6978 - val_loss: 0.9217 - val_accuracy: 0.6950 - lr: 1.0000e-07
Epoch 83/100
600/600 [==============================] - 7s 11ms/step - loss: 0.9238 -
accuracy: 0.6961 - val_loss: 0.9224 - val_accuracy: 0.6946 - lr: 1.0000e-07
Epoch 84/100
600/600 [==============================] - 9s 14ms/step - loss: 0.9359 -
accuracy: 0.6941 - val_loss: 0.9224 - val_accuracy: 0.6940 - lr: 1.0000e-07
Epoch 85/100
600/600 [==============================] - 8s 13ms/step - loss: 0.9324 -
accuracy: 0.6933 - val_loss: 0.9224 - val_accuracy: 0.6946 - lr: 1.0000e-07
Epoch 86/100
600/600 [==============================] - 6s 11ms/step - loss: 0.9343 -
accuracy: 0.6946 - val_loss: 0.9229 - val_accuracy: 0.6953 - lr: 1.0000e-07
```

```
Epoch 87/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9298 -
accuracy: 0.6958 - val_loss: 0.9224 - val_accuracy: 0.6952 - lr: 1.0000e-07
Epoch 88/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9244 -
accuracy: 0.6987 - val_loss: 0.9229 - val_accuracy: 0.6955 - lr: 1.0000e-07
Epoch 89/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9252 -
accuracy: 0.6977 - val_loss: 0.9226 - val_accuracy: 0.6949 - lr: 1.0000e-07
Epoch 90/100
600/600 [==============================] - 7s 12ms/step - loss: 0.9285 -
accuracy: 0.6954 - val_loss: 0.9222 - val_accuracy: 0.6944 - lr: 1.0000e-07
Epoch 91/100
600/600 [==============================] - 8s 13ms/step - loss: 0.9271 -
accuracy: 0.6995 - val_loss: 0.9226 - val_accuracy: 0.6953 - lr: 1.0000e-07
Epoch 92/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9258 -
accuracy: 0.6985 - val_loss: 0.9232 - val_accuracy: 0.6951 - lr: 1.0000e-07
Epoch 93/100
600/600 [==============================] - 7s 11ms/step - loss: 0.9297 -
accuracy: 0.6968 - val_loss: 0.9222 - val_accuracy: 0.6947 - lr: 1.0000e-07
Epoch 94/100
600/600 [==============================] - 8s 13ms/step - loss: 0.9337 -
accuracy: 0.6949 - val_loss: 0.9226 - val_accuracy: 0.6963 - lr: 1.0000e-07
Epoch 95/100
600/600 [==============================] - 9s 14ms/step - loss: 0.9271 -
accuracy: 0.6986 - val_loss: 0.9224 - val_accuracy: 0.6952 - lr: 1.0000e-07
Epoch 96/100
600/600 [==============================] - 8s 14ms/step - loss: 0.9260 -
accuracy: 0.6945 - val_loss: 0.9220 - val_accuracy: 0.6939 - lr: 1.0000e-07
Epoch 97/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9268 -
accuracy: 0.6941 - val_loss: 0.9221 - val_accuracy: 0.6950 - lr: 1.0000e-07
Epoch 98/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9286 -
accuracy: 0.6977 - val_loss: 0.9226 - val_accuracy: 0.6964 - lr: 1.0000e-07
Epoch 99/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9311 -
accuracy: 0.6963 - val_loss: 0.9227 - val_accuracy: 0.6955 - lr: 1.0000e-07
Epoch 100/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9306 -
accuracy: 0.6977 - val_loss: 0.9228 - val_accuracy: 0.6945 - lr: 1.0000e-07
```




```
375/375 [==============================] - 3s 5ms/step - loss: 0.9445 -
accuracy: 0.6931
Test accuracy: 0.693
```

2.5 Set 5: Removing noise using KNN (inspiration from this article). - DOES NOT WORK.

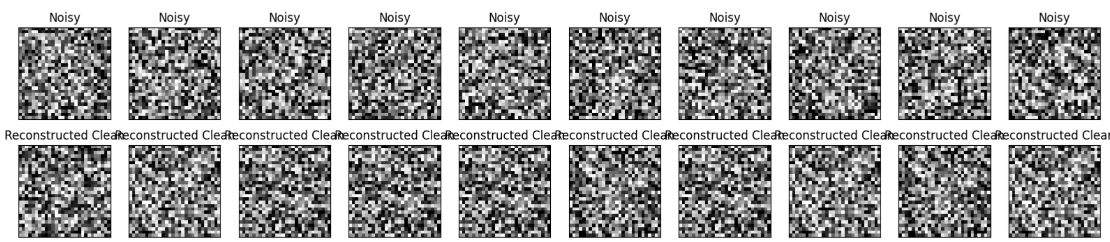
[ ]:
```
from sklearn.neighbors import KNeighborsClassifier
# reshape dataset
knn_X_train = X_train.reshape(-1, 784) # .reshape()
knn_X_test = X_test.reshape(-1, 784) # .reshape()
knn_y_train = y_train.reshape(-1) # .reshape()
knn_y_test = y_test.reshape(-1) # .reshape()

knn_clf = KNeighborsClassifier()
knn_clf.fit(np.asarray(knn_X_train), knn_y_train)
predicted_labels = knn_clf.predict(np.asarray(knn_X_test))
```

[ ]:
```
KNeighborsClassifier()
```

[ ]:
```
# Create an array to store the reconstructed clean images
reconstructed_images = np.zeros_like(knn_X_test)

# Loop through the noisy test images and select the corresponding clean images
for i in range(len(knn_X_test)):
    label = predicted_labels[i]
    clean_image = knn_X_train[knn_y_train == label][0]  # Assuming the first␣
↪occurrence of the label is the clean image
```

```
    reconstructed_images[i] = clean_image
```

[ ]:
```
n = 10  # Number of images to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display noisy image
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(knn_X_test[i].reshape(28, 28), cmap='gray')
    plt.title('Noisy')
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstructed clean image
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(reconstructed_images[i].reshape(28, 28), cmap='gray')
    plt.title('Reconstructed Clean')
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()
```



2.6 Set 6: Training with clean MNIST dataset together.

Gives 9.9% accuracy

[ ]:
```
# Load training images and labels for your CSCE636 project
csce636_train_images = pickle.load(open('drive/MyDrive/DeepLearningCSCE636/
 ↪636_project1_train_images', 'rb'))
csce636_train_labels = pickle.load(open('drive/MyDrive/DeepLearningCSCE636/
 ↪636_project1_train_labels', 'rb'))

# Normalize the pixel values of the images to a range between 0 and 1
csce636_train_images = csce636_train_images.astype('float') / 255.

# Reshape the images to the required format (assuming a 28x28 size with 1␣
 ↪channel)
```

```python
csce636_train_images = csce636_train_images.reshape((csce636_train_images.shape[0], 28, 28, 1))

# Split the CSCE636 training data into training and testing sets
csce636_X_train, csce636_X_test, csce636_y_train, csce636_y_test = train_test_split(csce636_train_images, csce636_train_labels, test_size=0.2, random_state=12)

# Load the MNIST dataset
(mnist_X_train, mnist_y_train), (mnist_X_test, mnist_y_test) = mnist.load_data()

# Normalize the pixel values of the MNIST images to a range between 0 and 1
mnist_X_train = mnist_X_train.astype('float') / 255.
mnist_X_test = mnist_X_test.astype('float') / 255.

# Reshape the MNIST images to the required format
mnist_X_train = np.reshape(mnist_X_train, (60000, 28, 28, 1))
mnist_X_test = np.reshape(mnist_X_test, (10000, 28, 28, 1))

# Combine the CSCE636 and MNIST training data
combined_X_train = np.vstack((csce636_X_train, mnist_X_train))
combined_y_train = np.hstack((csce636_y_train, mnist_y_train))

# Shuffle the combined dataset
combined_X_train, combined_y_train = shuffle(combined_X_train, combined_y_train, random_state=42)

# Create a new array for the combined dataset
c_new_X_train = np.array(combined_X_train)
c_new_X_test = np.array(csce636_X_test)

# Pad the images with 2 pixels on all sides to make them 32x32 as mentioned in the paper
c_new_X_train = np.pad(c_new_X_train, ((0, 0), (2, 2), (2, 2), (0, 0)), 'constant')
c_new_X_test = np.pad(c_new_X_test, ((0, 0), (2, 2), (2, 2), (0, 0)), 'constant')

# Standardize the pixel values of the images
mean_px = new_X_train.mean().astype(np.float32)
std_px = new_X_train.std().astype(np.float32)
c_new_X_train = (c_new_X_train - mean_px) / (std_px)

mean_px = new_X_test.mean().astype(np.float32)
std_px = new_X_test.std().astype(np.float32)
c_new_X_test = (c_new_X_test - mean_px) / (std_px)
```

```
# One-hot encode the labels for the combined dataset and the CSCE636 test set
c_new_y_train = to_categorical(combined_y_train, num_classes=10)
c_new_y_test = to_categorical(csce636_y_test, num_classes=10)
```

[ ]:

```
# define input layer
c_new_inputs = keras.Input(shape=(32, 32, 1))

# add layers to model
x = layers.Conv2D(filters = 32, kernel_size = 5, strides = 1, activation =␣
↪'relu', input_shape = (32,32,1), kernel_regularizer=l2(0.0005))(c_new_inputs)
x = layers.Conv2D(filters = 32, kernel_size = 5, strides = 1, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size = 2, strides = 2)(x)
x = layers.GaussianNoise(0.1)(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters = 64, kernel_size = 3, strides = 1, activation =␣
↪'relu', kernel_regularizer=l2(0.0005))(x)
x = layers.Conv2D(filters = 64, kernel_size = 3, strides = 1, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size = 2, strides = 2)(x)
x = layers.GaussianNoise(0.1)(x)
x = layers.Dropout(0.5)(x)
x = layers.Flatten()(x)
x = layers.Dense(units = 256, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dense(units = 128, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dense(units = 64, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dropout(0.5)(x)

# define output layer
c_new_outputs = layers.Dense(10, activation="softmax")(x)
# define model
model2 = keras.Model(inputs=c_new_inputs, outputs=c_new_outputs)
# compile model
model2.compile(optimizer = optimizers.Adam(),
  loss="categorical_crossentropy",
  metrics=["accuracy"])
# train model using validation split of 20% for 100 epochs
history2 = model2.fit(c_new_X_train,
```

```
                    c_new_y_train,
                    epochs=100,
                    validation_split = 0.2,
                    callbacks=[variable_learning_rate, model_checkpoint],
                    batch_size=64,
                    verbose=True)
plot_and_print(history2, model1, c_new_X_test, c_new_y_test)
```

```
Epoch 1/100
1350/1350 [==============================] - 24s 13ms/step - loss: 1.2357 -
accuracy: 0.5831 - val_loss: 0.9743 - val_accuracy: 0.6784 - lr: 0.0010
Epoch 2/100
1350/1350 [==============================] - 17s 13ms/step - loss: 1.0263 -
accuracy: 0.6517 - val_loss: 0.9122 - val_accuracy: 0.6880 - lr: 0.0010
Epoch 3/100
1350/1350 [==============================] - 18s 13ms/step - loss: 0.8968 -
accuracy: 0.7039 - val_loss: 0.6693 - val_accuracy: 0.7822 - lr: 0.0010
Epoch 4/100
1350/1350 [==============================] - 17s 13ms/step - loss: 0.7807 -
accuracy: 0.7508 - val_loss: 0.6497 - val_accuracy: 0.7856 - lr: 0.0010
Epoch 5/100
1350/1350 [==============================] - 16s 11ms/step - loss: 0.7405 -
accuracy: 0.7658 - val_loss: 0.6097 - val_accuracy: 0.8009 - lr: 0.0010
Epoch 6/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.7170 -
accuracy: 0.7735 - val_loss: 0.6509 - val_accuracy: 0.7881 - lr: 0.0010
Epoch 7/100
1350/1350 [==============================] - 22s 16ms/step - loss: 0.7004 -
accuracy: 0.7802 - val_loss: 0.5599 - val_accuracy: 0.8209 - lr: 0.0010
Epoch 8/100
1350/1350 [==============================] - 16s 12ms/step - loss: 0.6839 -
accuracy: 0.7869 - val_loss: 0.5539 - val_accuracy: 0.8238 - lr: 0.0010
Epoch 9/100
1350/1350 [==============================] - 16s 12ms/step - loss: 0.6690 -
accuracy: 0.7933 - val_loss: 0.5400 - val_accuracy: 0.8246 - lr: 0.0010
Epoch 10/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.6508 -
accuracy: 0.7999 - val_loss: 0.5845 - val_accuracy: 0.8120 - lr: 0.0010
Epoch 11/100
1350/1350 [==============================] - 16s 12ms/step - loss: 0.6346 -
accuracy: 0.8037 - val_loss: 0.5202 - val_accuracy: 0.8323 - lr: 0.0010
Epoch 12/100
1350/1350 [==============================] - 15s 11ms/step - loss: 0.6252 -
accuracy: 0.8066 - val_loss: 0.5139 - val_accuracy: 0.8394 - lr: 0.0010
Epoch 13/100
1350/1350 [==============================] - 13s 9ms/step - loss: 0.6167 -
accuracy: 0.8107 - val_loss: 0.5139 - val_accuracy: 0.8375 - lr: 0.0010
```

```
Epoch 14/100
1350/1350 [==============================] - 16s 12ms/step - loss: 0.6079 -
accuracy: 0.8125 - val_loss: 0.5005 - val_accuracy: 0.8391 - lr: 0.0010
Epoch 15/100
1350/1350 [==============================] - 13s 9ms/step - loss: 0.6028 -
accuracy: 0.8148 - val_loss: 0.5158 - val_accuracy: 0.8337 - lr: 0.0010
Epoch 16/100
1350/1350 [==============================] - 16s 12ms/step - loss: 0.5988 -
accuracy: 0.8170 - val_loss: 0.4978 - val_accuracy: 0.8409 - lr: 0.0010
Epoch 17/100
1350/1350 [==============================] - 17s 13ms/step - loss: 0.5896 -
accuracy: 0.8181 - val_loss: 0.4959 - val_accuracy: 0.8417 - lr: 0.0010
Epoch 18/100
1350/1350 [==============================] - 17s 13ms/step - loss: 0.5841 -
accuracy: 0.8203 - val_loss: 0.4941 - val_accuracy: 0.8411 - lr: 0.0010
Epoch 19/100
1350/1350 [==============================] - 18s 13ms/step - loss: 0.5878 -
accuracy: 0.8188 - val_loss: 0.4934 - val_accuracy: 0.8421 - lr: 0.0010
Epoch 20/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.5865 -
accuracy: 0.8197 - val_loss: 0.4985 - val_accuracy: 0.8406 - lr: 0.0010
Epoch 21/100
1350/1350 [==============================] - 16s 12ms/step - loss: 0.5752 -
accuracy: 0.8231 - val_loss: 0.4920 - val_accuracy: 0.8413 - lr: 0.0010
Epoch 22/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.5774 -
accuracy: 0.8222 - val_loss: 0.5157 - val_accuracy: 0.8357 - lr: 0.0010
Epoch 23/100
1350/1350 [==============================] - 13s 9ms/step - loss: 0.5663 -
accuracy: 0.8260 - val_loss: 0.5041 - val_accuracy: 0.8395 - lr: 0.0010
Epoch 24/100
1350/1350 [==============================] - 16s 12ms/step - loss: 0.5681 -
accuracy: 0.8247 - val_loss: 0.4886 - val_accuracy: 0.8444 - lr: 0.0010
Epoch 25/100
1350/1350 [==============================] - 13s 9ms/step - loss: 0.5631 -
accuracy: 0.8268 - val_loss: 0.5376 - val_accuracy: 0.8308 - lr: 0.0010
Epoch 26/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.5568 -
accuracy: 0.8279 - val_loss: 0.5046 - val_accuracy: 0.8374 - lr: 0.0010
Epoch 27/100
1350/1350 [==============================] - 15s 11ms/step - loss: 0.5573 -
accuracy: 0.8286 - val_loss: 0.4889 - val_accuracy: 0.8451 - lr: 0.0010
Epoch 28/100
1350/1350 [==============================] - 18s 13ms/step - loss: 0.5541 -
accuracy: 0.8298 - val_loss: 0.4867 - val_accuracy: 0.8445 - lr: 0.0010
Epoch 29/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.5495 -
accuracy: 0.8307 - val_loss: 0.4911 - val_accuracy: 0.8432 - lr: 0.0010
```

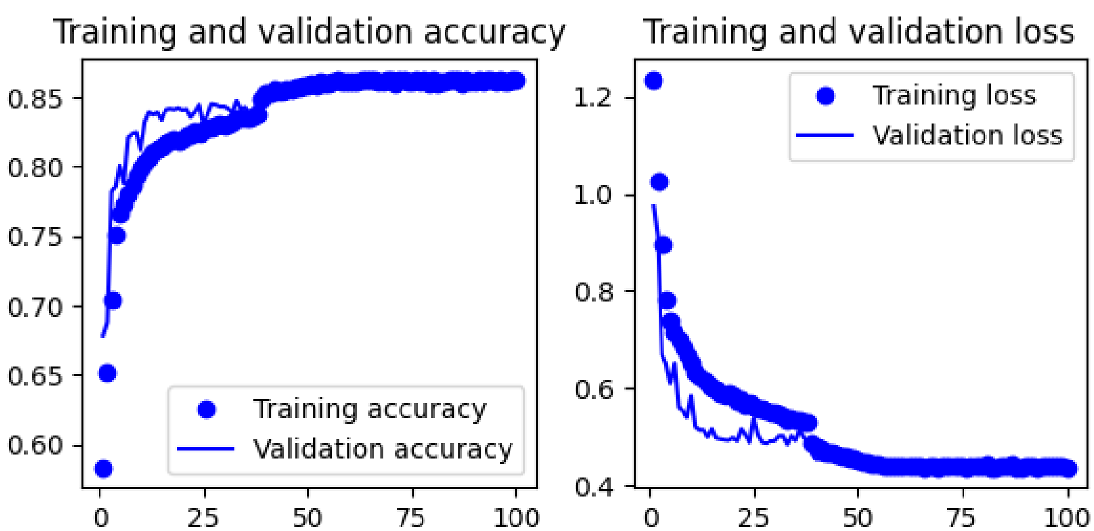
```
Epoch 30/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.5493 -
accuracy: 0.8296 - val_loss: 0.4920 - val_accuracy: 0.8432 - lr: 0.0010
Epoch 31/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.5469 -
accuracy: 0.8311 - val_loss: 0.5019 - val_accuracy: 0.8416 - lr: 0.0010
Epoch 32/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.5430 -
accuracy: 0.8319 - val_loss: 0.5020 - val_accuracy: 0.8387 - lr: 0.0010
Epoch 33/100
1350/1350 [==============================] - 16s 12ms/step - loss: 0.5346 -
accuracy: 0.8347 - val_loss: 0.4845 - val_accuracy: 0.8476 - lr: 0.0010
Epoch 34/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.5384 -
accuracy: 0.8345 - val_loss: 0.5027 - val_accuracy: 0.8415 - lr: 0.0010
Epoch 35/100
1350/1350 [==============================] - 15s 11ms/step - loss: 0.5347 -
accuracy: 0.8344 - val_loss: 0.4929 - val_accuracy: 0.8426 - lr: 0.0010
Epoch 36/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.5356 -
accuracy: 0.8350 - val_loss: 0.5140 - val_accuracy: 0.8385 - lr: 0.0010
Epoch 37/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.5321 -
accuracy: 0.8366 - val_loss: 0.4992 - val_accuracy: 0.8403 - lr: 0.0010
Epoch 38/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.5296 -
accuracy: 0.8376 - val_loss: 0.4878 - val_accuracy: 0.8452 - lr: 0.0010
Epoch 39/100
1350/1350 [==============================] - 16s 12ms/step - loss: 0.4882 -
accuracy: 0.8490 - val_loss: 0.4633 - val_accuracy: 0.8550 - lr: 1.0000e-04
Epoch 40/100
1350/1350 [==============================] - 16s 12ms/step - loss: 0.4779 -
accuracy: 0.8526 - val_loss: 0.4537 - val_accuracy: 0.8575 - lr: 1.0000e-04
Epoch 41/100
1350/1350 [==============================] - 14s 10ms/step - loss: 0.4751 -
accuracy: 0.8527 - val_loss: 0.4543 - val_accuracy: 0.8570 - lr: 1.0000e-04
Epoch 42/100
1350/1350 [==============================] - 13s 9ms/step - loss: 0.4684 -
accuracy: 0.8552 - val_loss: 0.4558 - val_accuracy: 0.8569 - lr: 1.0000e-04
Epoch 43/100
1350/1350 [==============================] - 17s 13ms/step - loss: 0.4671 -
accuracy: 0.8546 - val_loss: 0.4508 - val_accuracy: 0.8578 - lr: 1.0000e-04
Epoch 44/100
1350/1350 [==============================] - 14s 11ms/step - loss: 0.4671 -
accuracy: 0.8546 - val_loss: 0.4533 - val_accuracy: 0.8567 - lr: 1.0000e-04
Epoch 45/100
1350/1350 [==============================] - 16s 12ms/step - loss: 0.4619 -
accuracy: 0.8559 - val_loss: 0.4496 - val_accuracy: 0.8577 - lr: 1.0000e-04
```

```
Epoch 46/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4590 -
accuracy: 0.8552 - val_loss: 0.4528 - val_accuracy: 0.8575 - lr: 1.0000e-04
Epoch 47/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4589 -
accuracy: 0.8569 - val_loss: 0.4565 - val_accuracy: 0.8565 - lr: 1.0000e-04
Epoch 48/100
1350/1350 [==============================] - 16s 12ms/step - loss: 0.4568 -
accuracy: 0.8571 - val_loss: 0.4463 - val_accuracy: 0.8578 - lr: 1.0000e-04
Epoch 49/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4555 -
accuracy: 0.8584 - val_loss: 0.4464 - val_accuracy: 0.8580 - lr: 1.0000e-04
Epoch 50/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4536 -
accuracy: 0.8581 - val_loss: 0.4466 - val_accuracy: 0.8584 - lr: 1.0000e-04
Epoch 51/100
1350/1350 [==============================] - 15s 11ms/step - loss: 0.4488 -
accuracy: 0.8593 - val_loss: 0.4501 - val_accuracy: 0.8572 - lr: 1.0000e-04
Epoch 52/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4482 -
accuracy: 0.8582 - val_loss: 0.4502 - val_accuracy: 0.8581 - lr: 1.0000e-04
Epoch 53/100
1350/1350 [==============================] - 17s 13ms/step - loss: 0.4451 -
accuracy: 0.8610 - val_loss: 0.4556 - val_accuracy: 0.8568 - lr: 1.0000e-04
Epoch 54/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4444 -
accuracy: 0.8600 - val_loss: 0.4481 - val_accuracy: 0.8587 - lr: 1.0000e-05
Epoch 55/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4428 -
accuracy: 0.8604 - val_loss: 0.4478 - val_accuracy: 0.8581 - lr: 1.0000e-05
Epoch 56/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4420 -
accuracy: 0.8605 - val_loss: 0.4516 - val_accuracy: 0.8574 - lr: 1.0000e-05
Epoch 57/100
1350/1350 [==============================] - 13s 9ms/step - loss: 0.4412 -
accuracy: 0.8621 - val_loss: 0.4513 - val_accuracy: 0.8575 - lr: 1.0000e-05
Epoch 58/100
1350/1350 [==============================] - 14s 10ms/step - loss: 0.4395 -
accuracy: 0.8606 - val_loss: 0.4505 - val_accuracy: 0.8577 - lr: 1.0000e-05
Epoch 59/100
1350/1350 [==============================] - 14s 10ms/step - loss: 0.4393 -
accuracy: 0.8619 - val_loss: 0.4490 - val_accuracy: 0.8582 - lr: 1.0000e-06
Epoch 60/100
1350/1350 [==============================] - 14s 10ms/step - loss: 0.4384 -
accuracy: 0.8618 - val_loss: 0.4487 - val_accuracy: 0.8581 - lr: 1.0000e-06
Epoch 61/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4397 -
accuracy: 0.8615 - val_loss: 0.4482 - val_accuracy: 0.8581 - lr: 1.0000e-06
```

```
Epoch 62/100
1350/1350 [==============================] - 14s 10ms/step - loss: 0.4396 -
accuracy: 0.8616 - val_loss: 0.4483 - val_accuracy: 0.8581 - lr: 1.0000e-06
Epoch 63/100
1350/1350 [==============================] - 14s 10ms/step - loss: 0.4393 -
accuracy: 0.8625 - val_loss: 0.4489 - val_accuracy: 0.8578 - lr: 1.0000e-06
Epoch 64/100
1350/1350 [==============================] - 14s 10ms/step - loss: 0.4400 -
accuracy: 0.8619 - val_loss: 0.4486 - val_accuracy: 0.8581 - lr: 1.0000e-07
Epoch 65/100
1350/1350 [==============================] - 14s 10ms/step - loss: 0.4383 -
accuracy: 0.8625 - val_loss: 0.4484 - val_accuracy: 0.8581 - lr: 1.0000e-07
Epoch 66/100
1350/1350 [==============================] - 13s 9ms/step - loss: 0.4365 -
accuracy: 0.8627 - val_loss: 0.4485 - val_accuracy: 0.8576 - lr: 1.0000e-07
Epoch 67/100
1350/1350 [==============================] - 14s 10ms/step - loss: 0.4400 -
accuracy: 0.8613 - val_loss: 0.4492 - val_accuracy: 0.8581 - lr: 1.0000e-07
Epoch 68/100
1350/1350 [==============================] - 14s 11ms/step - loss: 0.4402 -
accuracy: 0.8616 - val_loss: 0.4489 - val_accuracy: 0.8582 - lr: 1.0000e-07
Epoch 69/100
1350/1350 [==============================] - 14s 10ms/step - loss: 0.4375 -
accuracy: 0.8628 - val_loss: 0.4484 - val_accuracy: 0.8582 - lr: 1.0000e-07
Epoch 70/100
1350/1350 [==============================] - 15s 11ms/step - loss: 0.4385 -
accuracy: 0.8622 - val_loss: 0.4487 - val_accuracy: 0.8581 - lr: 1.0000e-07
Epoch 71/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4388 -
accuracy: 0.8602 - val_loss: 0.4482 - val_accuracy: 0.8580 - lr: 1.0000e-07
Epoch 72/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4383 -
accuracy: 0.8620 - val_loss: 0.4482 - val_accuracy: 0.8584 - lr: 1.0000e-07
Epoch 73/100
1350/1350 [==============================] - 14s 11ms/step - loss: 0.4400 -
accuracy: 0.8620 - val_loss: 0.4481 - val_accuracy: 0.8582 - lr: 1.0000e-07
Epoch 74/100
1350/1350 [==============================] - 13s 9ms/step - loss: 0.4419 -
accuracy: 0.8610 - val_loss: 0.4485 - val_accuracy: 0.8580 - lr: 1.0000e-07
Epoch 75/100
1350/1350 [==============================] - 14s 10ms/step - loss: 0.4388 -
accuracy: 0.8617 - val_loss: 0.4481 - val_accuracy: 0.8583 - lr: 1.0000e-07
Epoch 76/100
1350/1350 [==============================] - 14s 10ms/step - loss: 0.4360 -
accuracy: 0.8632 - val_loss: 0.4478 - val_accuracy: 0.8582 - lr: 1.0000e-07
Epoch 77/100
1350/1350 [==============================] - 14s 10ms/step - loss: 0.4388 -
accuracy: 0.8617 - val_loss: 0.4479 - val_accuracy: 0.8581 - lr: 1.0000e-07
```

```
Epoch 78/100
1350/1350 [==============================] - 14s 11ms/step - loss: 0.4389 -
accuracy: 0.8628 - val_loss: 0.4487 - val_accuracy: 0.8581 - lr: 1.0000e-07
Epoch 79/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4414 -
accuracy: 0.8604 - val_loss: 0.4479 - val_accuracy: 0.8581 - lr: 1.0000e-07
Epoch 80/100
1350/1350 [==============================] - 14s 10ms/step - loss: 0.4403 -
accuracy: 0.8619 - val_loss: 0.4483 - val_accuracy: 0.8585 - lr: 1.0000e-07
Epoch 81/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4433 -
accuracy: 0.8603 - val_loss: 0.4485 - val_accuracy: 0.8583 - lr: 1.0000e-07
Epoch 82/100
1350/1350 [==============================] - 17s 12ms/step - loss: 0.4359 -
accuracy: 0.8618 - val_loss: 0.4481 - val_accuracy: 0.8582 - lr: 1.0000e-07
Epoch 83/100
1350/1350 [==============================] - 16s 12ms/step - loss: 0.4382 -
accuracy: 0.8616 - val_loss: 0.4479 - val_accuracy: 0.8585 - lr: 1.0000e-07
Epoch 84/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4395 -
accuracy: 0.8623 - val_loss: 0.4483 - val_accuracy: 0.8584 - lr: 1.0000e-07
Epoch 85/100
1350/1350 [==============================] - 14s 10ms/step - loss: 0.4396 -
accuracy: 0.8625 - val_loss: 0.4482 - val_accuracy: 0.8581 - lr: 1.0000e-07
Epoch 86/100
1350/1350 [==============================] - 14s 10ms/step - loss: 0.4402 -
accuracy: 0.8621 - val_loss: 0.4484 - val_accuracy: 0.8581 - lr: 1.0000e-07
Epoch 87/100
1350/1350 [==============================] - 15s 11ms/step - loss: 0.4435 -
accuracy: 0.8604 - val_loss: 0.4487 - val_accuracy: 0.8581 - lr: 1.0000e-07
Epoch 88/100
1350/1350 [==============================] - 25s 18ms/step - loss: 0.4376 -
accuracy: 0.8622 - val_loss: 0.4480 - val_accuracy: 0.8583 - lr: 1.0000e-07
Epoch 89/100
1350/1350 [==============================] - 21s 15ms/step - loss: 0.4377 -
accuracy: 0.8614 - val_loss: 0.4485 - val_accuracy: 0.8581 - lr: 1.0000e-07
Epoch 90/100
1350/1350 [==============================] - 20s 14ms/step - loss: 0.4421 -
accuracy: 0.8613 - val_loss: 0.4480 - val_accuracy: 0.8584 - lr: 1.0000e-07
Epoch 91/100
1350/1350 [==============================] - 15s 11ms/step - loss: 0.4390 -
accuracy: 0.8625 - val_loss: 0.4481 - val_accuracy: 0.8583 - lr: 1.0000e-07
Epoch 92/100
1350/1350 [==============================] - 19s 14ms/step - loss: 0.4417 -
accuracy: 0.8613 - val_loss: 0.4483 - val_accuracy: 0.8581 - lr: 1.0000e-07
Epoch 93/100
1350/1350 [==============================] - 25s 18ms/step - loss: 0.4382 -
accuracy: 0.8620 - val_loss: 0.4487 - val_accuracy: 0.8585 - lr: 1.0000e-07
```

```
Epoch 94/100
1350/1350 [==============================] - 20s 15ms/step - loss: 0.4397 -
accuracy: 0.8619 - val_loss: 0.4484 - val_accuracy: 0.8583 - lr: 1.0000e-07
Epoch 95/100
1350/1350 [==============================] - 20s 15ms/step - loss: 0.4389 -
accuracy: 0.8607 - val_loss: 0.4485 - val_accuracy: 0.8585 - lr: 1.0000e-07
Epoch 96/100
1350/1350 [==============================] - 19s 14ms/step - loss: 0.4385 -
accuracy: 0.8620 - val_loss: 0.4482 - val_accuracy: 0.8581 - lr: 1.0000e-07
Epoch 97/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4417 -
accuracy: 0.8614 - val_loss: 0.4486 - val_accuracy: 0.8584 - lr: 1.0000e-07
Epoch 98/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4403 -
accuracy: 0.8608 - val_loss: 0.4482 - val_accuracy: 0.8588 - lr: 1.0000e-07
Epoch 99/100
1350/1350 [==============================] - 13s 10ms/step - loss: 0.4389 -
accuracy: 0.8621 - val_loss: 0.4480 - val_accuracy: 0.8586 - lr: 1.0000e-07
Epoch 100/100
1350/1350 [==============================] - 14s 11ms/step - loss: 0.4372 -
accuracy: 0.8623 - val_loss: 0.4482 - val_accuracy: 0.8580 - lr: 1.0000e-07
```




```
375/375 [==============================] - 1s 4ms/step - loss: 2.7724 -
accuracy: 0.0989
Test accuracy: 0.099
```

3 Final Model

After all different experiments and results obtained, I have decided to consider the following as my final model.

- Preprocessing:
  1. Reshape each image
  2. Convert to float and normalize
  3. Convert to numpy array
  4. Add padding to make 32x32 size
  5. Standardize each pixel using mean and standard deviation
  6. One hot encoding for labels (10 classes)
- Deep Learning Model:

1. Convolutional layer with 32 filters, a kernel size of 5, ReLU activation, and L2 regularization for an input shape of 32x32x1.
2. Convolutional layer with 32 filters, a kernel size of 5, and no bias terms.
3. Add batch normalization to the previous layer.
4. Apply the ReLU activation function to the output of the batch normalization.
5. Implement max-pooling with a pool size of 2x2 and a stride of 2.
6. Apply dropout regularization with a rate of 0.5.
7. Add another convolutional layer with 64 filters, a kernel size of 3, ReLU activation, and L2 regularization.
8. Create another convolutional layer with 64 filters, a kernel size of 3, and no bias terms.
9. Add batch normalization to the previous layer.
10. Apply the ReLU activation function to the output of the batch normalization.
11. Implement max-pooling with a pool size of 2x2 and a stride of 2.
12. Apply dropout regularization with a rate of 0.5.
13. Flatten the output of the previous layers.
14. Create a dense (fully connected) layer with 256 units and no bias terms.
15. Add batch normalization to the dense layer.
16. Apply the ReLU activation function to the output of the batch normalization.
17. Create another dense layer with 128 units and no bias terms.
18. Add batch normalization to the dense layer.
19. Apply the ReLU activation function to the output of the batch normalization.
20. Create a final dense layer with 64 units and no bias terms.
21. Add batch normalization to the dense layer.
22. Apply the ReLU activation function to the output of the batch normalization.
23. Apply dropout regularization with a rate of 0.5.
24. Add final output layer as a dense layer with 10 units.

- Use categorical_entropy as loss function.
- Use Adam optimizer.
- Train for 100 epochs and save best weights according validation loss values.
- Use batch size of 64 and cross validation split of 0.2.

Preprocessing input

```
[ ]: # Convert the datasets to NumPy arrays
X_train = np.array(X_train)
X_test = np.array(X_test)

# Pad the images with 2 pixels on all sides to make them 32x32
X_train = np.pad(X_train, ((0, 0), (2, 2), (2, 2), (0, 0)), 'constant')
X_test = np.pad(X_test, ((0, 0), (2, 2), (2, 2), (0, 0)), 'constant')

# Standardize the pixel values of the images
mean_px = X_train.mean().astype(np.float32)
std_px = X_train.std().astype(np.float32)
X_train = (X_train - mean_px) / (std_px)

mean_px = X_test.mean().astype(np.float32)
std_px = X_test.std().astype(np.float32)
X_test = (X_test - mean_px) / (std_px)

# One-hot encode the labels for training and testing datasets
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

Define layers of model

```
[ ]: # define input layer
inputs = keras.Input(shape=(32, 32, 1))

# define and add layers to model
x = layers.Conv2D(filters = 32, kernel_size = 5, strides = 1, activation =␣
 ↪'relu', input_shape = (32,32,1), kernel_regularizer=l2(0.0005))(inputs)
x = layers.Conv2D(filters = 32, kernel_size = 5, strides = 1, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size = 2, strides = 2)(x)
x = layers.Dropout(0.5)(x)
x = layers.Conv2D(filters = 64, kernel_size = 3, strides = 1, activation =␣
 ↪'relu', kernel_regularizer=l2(0.0005))(x)
x = layers.Conv2D(filters = 64, kernel_size = 3, strides = 1, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size = 2, strides = 2)(x)
x = layers.Dropout(0.5)(x)
x = layers.Flatten()(x)
x = layers.Dense(units = 256, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dense(units = 128, use_bias=False)(x)
x = layers.BatchNormalization()(x)
```

```python
x = layers.Activation('relu')(x)
x = layers.Dense(units = 64, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.Dropout(0.5)(x)

# define output layer
outputs = layers.Dense(10, activation="softmax")(x)
# define model
final_model = keras.Model(inputs=inputs, outputs=outputs)
# compile model
final_model.compile(optimizer = optimizers.Adam(),
  loss="categorical_crossentropy",
  metrics=["accuracy"])

# train model using validation split of 20% for 100 epochs
history_final = final_model.fit(X_train,
                    y_train,
                    epochs=100,
                    validation_split = 0.2,
                    callbacks=[variable_learning_rate, model_checkpoint],
                    batch_size=64,
                    verbose=True)
plot_and_print(history_final, final_model, X_test, y_test)
```

```
Epoch 1/100
600/600 [==============================] - 23s 15ms/step - loss: 2.3713 -
accuracy: 0.1768 - val_loss: 2.0575 - val_accuracy: 0.2745 - lr: 0.0010
Epoch 2/100
600/600 [==============================] - 9s 15ms/step - loss: 2.0764 -
accuracy: 0.2738 - val_loss: 2.0088 - val_accuracy: 0.2979 - lr: 0.0010
Epoch 3/100
600/600 [==============================] - 8s 13ms/step - loss: 2.0072 -
accuracy: 0.3059 - val_loss: 1.8817 - val_accuracy: 0.3495 - lr: 0.0010
Epoch 4/100
600/600 [==============================] - 6s 11ms/step - loss: 1.8139 -
accuracy: 0.3846 - val_loss: 1.9748 - val_accuracy: 0.3298 - lr: 0.0010
Epoch 5/100
600/600 [==============================] - 9s 16ms/step - loss: 1.5433 -
accuracy: 0.4902 - val_loss: 1.2994 - val_accuracy: 0.5707 - lr: 0.0010
Epoch 6/100
600/600 [==============================] - 9s 15ms/step - loss: 1.4137 -
accuracy: 0.5351 - val_loss: 1.2273 - val_accuracy: 0.5895 - lr: 0.0010
Epoch 7/100
600/600 [==============================] - 8s 14ms/step - loss: 1.3604 -
accuracy: 0.5579 - val_loss: 1.2264 - val_accuracy: 0.5839 - lr: 0.0010
Epoch 8/100
```

```
600/600 [==============================] - 8s 13ms/step - loss: 1.3208 -
accuracy: 0.5688 - val_loss: 1.1414 - val_accuracy: 0.6125 - lr: 0.0010
Epoch 9/100
600/600 [==============================] - 8s 13ms/step - loss: 1.2856 -
accuracy: 0.5816 - val_loss: 1.0892 - val_accuracy: 0.6276 - lr: 0.0010
Epoch 10/100
600/600 [==============================] - 5s 9ms/step - loss: 1.2699 -
accuracy: 0.5891 - val_loss: 1.1456 - val_accuracy: 0.6149 - lr: 0.0010
Epoch 11/100
600/600 [==============================] - 8s 14ms/step - loss: 1.2493 -
accuracy: 0.5968 - val_loss: 1.0869 - val_accuracy: 0.6353 - lr: 0.0010
Epoch 12/100
600/600 [==============================] - 5s 9ms/step - loss: 1.2309 -
accuracy: 0.5992 - val_loss: 1.1013 - val_accuracy: 0.6291 - lr: 0.0010
Epoch 13/100
600/600 [==============================] - 8s 14ms/step - loss: 1.2193 -
accuracy: 0.6051 - val_loss: 1.0556 - val_accuracy: 0.6415 - lr: 0.0010
Epoch 14/100
600/600 [==============================] - 6s 10ms/step - loss: 1.2061 -
accuracy: 0.6114 - val_loss: 1.0790 - val_accuracy: 0.6415 - lr: 0.0010
Epoch 15/100
600/600 [==============================] - 8s 14ms/step - loss: 1.2060 -
accuracy: 0.6093 - val_loss: 1.0404 - val_accuracy: 0.6508 - lr: 0.0010
Epoch 16/100
600/600 [==============================] - 6s 9ms/step - loss: 1.1851 -
accuracy: 0.6154 - val_loss: 1.0517 - val_accuracy: 0.6454 - lr: 0.0010
Epoch 17/100
600/600 [==============================] - 6s 10ms/step - loss: 1.1747 -
accuracy: 0.6219 - val_loss: 1.0537 - val_accuracy: 0.6366 - lr: 0.0010
Epoch 18/100
600/600 [==============================] - 5s 9ms/step - loss: 1.1651 -
accuracy: 0.6239 - val_loss: 1.1158 - val_accuracy: 0.6257 - lr: 0.0010
Epoch 19/100
600/600 [==============================] - 6s 9ms/step - loss: 1.1599 -
accuracy: 0.6258 - val_loss: 1.0651 - val_accuracy: 0.6389 - lr: 0.0010
Epoch 20/100
600/600 [==============================] - 5s 9ms/step - loss: 1.1574 -
accuracy: 0.6255 - val_loss: 1.0571 - val_accuracy: 0.6468 - lr: 0.0010
Epoch 21/100
600/600 [==============================] - 8s 13ms/step - loss: 1.0717 -
accuracy: 0.6539 - val_loss: 0.9710 - val_accuracy: 0.6754 - lr: 1.0000e-04
Epoch 22/100
600/600 [==============================] - 8s 14ms/step - loss: 1.0539 -
accuracy: 0.6585 - val_loss: 0.9493 - val_accuracy: 0.6790 - lr: 1.0000e-04
Epoch 23/100
600/600 [==============================] - 5s 9ms/step - loss: 1.0462 -
accuracy: 0.6616 - val_loss: 0.9541 - val_accuracy: 0.6798 - lr: 1.0000e-04
Epoch 24/100
```

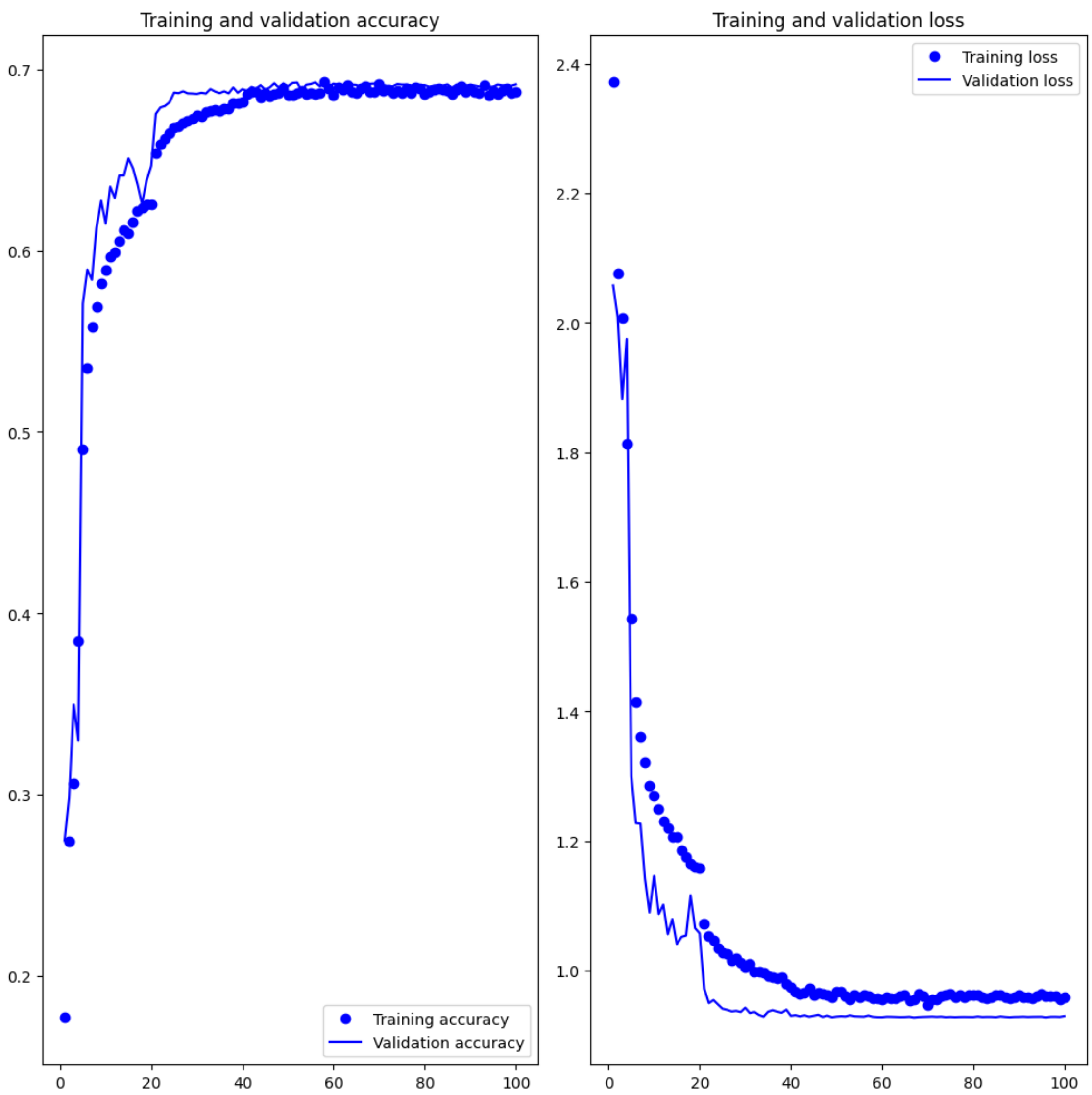
```
600/600 [==============================] - 8s 14ms/step - loss: 1.0336 -
accuracy: 0.6651 - val_loss: 0.9473 - val_accuracy: 0.6818 - lr: 1.0000e-04
Epoch 25/100
600/600 [==============================] - 9s 15ms/step - loss: 1.0265 -
accuracy: 0.6677 - val_loss: 0.9407 - val_accuracy: 0.6872 - lr: 1.0000e-04
Epoch 26/100
600/600 [==============================] - 8s 13ms/step - loss: 1.0249 -
accuracy: 0.6684 - val_loss: 0.9389 - val_accuracy: 0.6869 - lr: 1.0000e-04
Epoch 27/100
600/600 [==============================] - 10s 17ms/step - loss: 1.0156 -
accuracy: 0.6705 - val_loss: 0.9365 - val_accuracy: 0.6879 - lr: 1.0000e-04
Epoch 28/100
600/600 [==============================] - 5s 9ms/step - loss: 1.0182 -
accuracy: 0.6714 - val_loss: 0.9373 - val_accuracy: 0.6868 - lr: 1.0000e-04
Epoch 29/100
600/600 [==============================] - 8s 14ms/step - loss: 1.0111 -
accuracy: 0.6727 - val_loss: 0.9355 - val_accuracy: 0.6867 - lr: 1.0000e-04
Epoch 30/100
600/600 [==============================] - 5s 9ms/step - loss: 1.0055 -
accuracy: 0.6745 - val_loss: 0.9422 - val_accuracy: 0.6865 - lr: 1.0000e-04
Epoch 31/100
600/600 [==============================] - 9s 15ms/step - loss: 1.0106 -
accuracy: 0.6742 - val_loss: 0.9340 - val_accuracy: 0.6871 - lr: 1.0000e-04
Epoch 32/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9989 -
accuracy: 0.6768 - val_loss: 0.9357 - val_accuracy: 0.6867 - lr: 1.0000e-04
Epoch 33/100
600/600 [==============================] - 9s 15ms/step - loss: 0.9985 -
accuracy: 0.6774 - val_loss: 0.9311 - val_accuracy: 0.6892 - lr: 1.0000e-04
Epoch 34/100
600/600 [==============================] - 7s 12ms/step - loss: 0.9960 -
accuracy: 0.6777 - val_loss: 0.9285 - val_accuracy: 0.6878 - lr: 1.0000e-04
Epoch 35/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9909 -
accuracy: 0.6771 - val_loss: 0.9361 - val_accuracy: 0.6869 - lr: 1.0000e-04
Epoch 36/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9895 -
accuracy: 0.6785 - val_loss: 0.9384 - val_accuracy: 0.6879 - lr: 1.0000e-04
Epoch 37/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9880 -
accuracy: 0.6781 - val_loss: 0.9362 - val_accuracy: 0.6868 - lr: 1.0000e-04
Epoch 38/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9890 -
accuracy: 0.6816 - val_loss: 0.9345 - val_accuracy: 0.6900 - lr: 1.0000e-04
Epoch 39/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9786 -
accuracy: 0.6815 - val_loss: 0.9392 - val_accuracy: 0.6871 - lr: 1.0000e-04
Epoch 40/100
```

```
600/600 [==============================] - 6s 10ms/step - loss: 0.9738 -
accuracy: 0.6818 - val_loss: 0.9297 - val_accuracy: 0.6891 - lr: 1.0000e-05
Epoch 41/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9677 -
accuracy: 0.6865 - val_loss: 0.9307 - val_accuracy: 0.6882 - lr: 1.0000e-05
Epoch 42/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9640 -
accuracy: 0.6879 - val_loss: 0.9289 - val_accuracy: 0.6903 - lr: 1.0000e-05
Epoch 43/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9659 -
accuracy: 0.6878 - val_loss: 0.9305 - val_accuracy: 0.6894 - lr: 1.0000e-05
Epoch 44/100
600/600 [==============================] - 8s 14ms/step - loss: 0.9725 -
accuracy: 0.6844 - val_loss: 0.9284 - val_accuracy: 0.6913 - lr: 1.0000e-05
Epoch 45/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9616 -
accuracy: 0.6868 - val_loss: 0.9298 - val_accuracy: 0.6889 - lr: 1.0000e-05
Epoch 46/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9648 -
accuracy: 0.6852 - val_loss: 0.9314 - val_accuracy: 0.6900 - lr: 1.0000e-05
Epoch 47/100
600/600 [==============================] - 8s 14ms/step - loss: 0.9633 -
accuracy: 0.6865 - val_loss: 0.9281 - val_accuracy: 0.6923 - lr: 1.0000e-05
Epoch 48/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9626 -
accuracy: 0.6870 - val_loss: 0.9301 - val_accuracy: 0.6899 - lr: 1.0000e-05
Epoch 49/100
600/600 [==============================] - 8s 14ms/step - loss: 0.9584 -
accuracy: 0.6893 - val_loss: 0.9275 - val_accuracy: 0.6922 - lr: 1.0000e-05
Epoch 50/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9680 -
accuracy: 0.6857 - val_loss: 0.9286 - val_accuracy: 0.6906 - lr: 1.0000e-05
Epoch 51/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9662 -
accuracy: 0.6855 - val_loss: 0.9291 - val_accuracy: 0.6925 - lr: 1.0000e-05
Epoch 52/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9609 -
accuracy: 0.6870 - val_loss: 0.9287 - val_accuracy: 0.6927 - lr: 1.0000e-05
Epoch 53/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9553 -
accuracy: 0.6880 - val_loss: 0.9305 - val_accuracy: 0.6884 - lr: 1.0000e-05
Epoch 54/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9618 -
accuracy: 0.6863 - val_loss: 0.9290 - val_accuracy: 0.6915 - lr: 1.0000e-05
Epoch 55/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9592 -
accuracy: 0.6870 - val_loss: 0.9287 - val_accuracy: 0.6919 - lr: 1.0000e-06
Epoch 56/100
```

```
600/600 [==============================] - 6s 10ms/step - loss: 0.9616 -
accuracy: 0.6862 - val_loss: 0.9284 - val_accuracy: 0.6928 - lr: 1.0000e-06
Epoch 57/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9602 -
accuracy: 0.6871 - val_loss: 0.9302 - val_accuracy: 0.6906 - lr: 1.0000e-06
Epoch 58/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9563 -
accuracy: 0.6929 - val_loss: 0.9282 - val_accuracy: 0.6927 - lr: 1.0000e-06
Epoch 59/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9573 -
accuracy: 0.6894 - val_loss: 0.9276 - val_accuracy: 0.6905 - lr: 1.0000e-06
Epoch 60/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9546 -
accuracy: 0.6860 - val_loss: 0.9275 - val_accuracy: 0.6921 - lr: 1.0000e-07
Epoch 61/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9583 -
accuracy: 0.6901 - val_loss: 0.9283 - val_accuracy: 0.6915 - lr: 1.0000e-07
Epoch 62/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9572 -
accuracy: 0.6888 - val_loss: 0.9283 - val_accuracy: 0.6905 - lr: 1.0000e-07
Epoch 63/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9568 -
accuracy: 0.6913 - val_loss: 0.9280 - val_accuracy: 0.6905 - lr: 1.0000e-07
Epoch 64/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9607 -
accuracy: 0.6873 - val_loss: 0.9277 - val_accuracy: 0.6916 - lr: 1.0000e-07
Epoch 65/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9620 -
accuracy: 0.6872 - val_loss: 0.9278 - val_accuracy: 0.6914 - lr: 1.0000e-08
Epoch 66/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9532 -
accuracy: 0.6894 - val_loss: 0.9283 - val_accuracy: 0.6901 - lr: 1.0000e-08
Epoch 67/100
600/600 [==============================] - 8s 13ms/step - loss: 0.9553 -
accuracy: 0.6908 - val_loss: 0.9271 - val_accuracy: 0.6918 - lr: 1.0000e-08
Epoch 68/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9629 -
accuracy: 0.6874 - val_loss: 0.9277 - val_accuracy: 0.6920 - lr: 1.0000e-08
Epoch 69/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9598 -
accuracy: 0.6874 - val_loss: 0.9281 - val_accuracy: 0.6922 - lr: 1.0000e-08
Epoch 70/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9464 -
accuracy: 0.6918 - val_loss: 0.9283 - val_accuracy: 0.6910 - lr: 1.0000e-08
Epoch 71/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9552 -
accuracy: 0.6882 - val_loss: 0.9286 - val_accuracy: 0.6913 - lr: 1.0000e-08
Epoch 72/100
```

```
600/600 [==============================] - 5s 9ms/step - loss: 0.9548 -
accuracy: 0.6886 - val_loss: 0.9281 - val_accuracy: 0.6907 - lr: 1.0000e-08
Epoch 73/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9595 -
accuracy: 0.6868 - val_loss: 0.9285 - val_accuracy: 0.6906 - lr: 1.0000e-09
Epoch 74/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9612 -
accuracy: 0.6883 - val_loss: 0.9276 - val_accuracy: 0.6919 - lr: 1.0000e-09
Epoch 75/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9640 -
accuracy: 0.6868 - val_loss: 0.9280 - val_accuracy: 0.6916 - lr: 1.0000e-09
Epoch 76/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9579 -
accuracy: 0.6893 - val_loss: 0.9276 - val_accuracy: 0.6916 - lr: 1.0000e-09
Epoch 77/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9615 -
accuracy: 0.6867 - val_loss: 0.9279 - val_accuracy: 0.6911 - lr: 1.0000e-09
Epoch 78/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9589 -
accuracy: 0.6899 - val_loss: 0.9280 - val_accuracy: 0.6903 - lr: 1.0000e-10
Epoch 79/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9613 -
accuracy: 0.6893 - val_loss: 0.9279 - val_accuracy: 0.6911 - lr: 1.0000e-10
Epoch 80/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9620 -
accuracy: 0.6864 - val_loss: 0.9278 - val_accuracy: 0.6913 - lr: 1.0000e-10
Epoch 81/100
600/600 [==============================] - 7s 11ms/step - loss: 0.9626 -
accuracy: 0.6878 - val_loss: 0.9287 - val_accuracy: 0.6904 - lr: 1.0000e-10
Epoch 82/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9585 -
accuracy: 0.6890 - val_loss: 0.9279 - val_accuracy: 0.6909 - lr: 1.0000e-10
Epoch 83/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9574 -
accuracy: 0.6898 - val_loss: 0.9281 - val_accuracy: 0.6911 - lr: 1.0000e-11
Epoch 84/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9577 -
accuracy: 0.6886 - val_loss: 0.9281 - val_accuracy: 0.6909 - lr: 1.0000e-11
Epoch 85/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9611 -
accuracy: 0.6885 - val_loss: 0.9276 - val_accuracy: 0.6917 - lr: 1.0000e-11
Epoch 86/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9624 -
accuracy: 0.6866 - val_loss: 0.9288 - val_accuracy: 0.6899 - lr: 1.0000e-11
Epoch 87/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9590 -
accuracy: 0.6891 - val_loss: 0.9280 - val_accuracy: 0.6916 - lr: 1.0000e-11
Epoch 88/100
```

```
600/600 [==============================] - 6s 9ms/step - loss: 0.9563 -
accuracy: 0.6904 - val_loss: 0.9275 - val_accuracy: 0.6909 - lr: 1.0000e-12
Epoch 89/100
600/600 [==============================] - 6s 9ms/step - loss: 0.9586 -
accuracy: 0.6877 - val_loss: 0.9280 - val_accuracy: 0.6910 - lr: 1.0000e-12
Epoch 90/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9618 -
accuracy: 0.6892 - val_loss: 0.9281 - val_accuracy: 0.6911 - lr: 1.0000e-12
Epoch 91/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9585 -
accuracy: 0.6877 - val_loss: 0.9284 - val_accuracy: 0.6910 - lr: 1.0000e-12
Epoch 92/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9590 -
accuracy: 0.6872 - val_loss: 0.9280 - val_accuracy: 0.6905 - lr: 1.0000e-12
Epoch 93/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9569 -
accuracy: 0.6915 - val_loss: 0.9282 - val_accuracy: 0.6901 - lr: 1.0000e-13
Epoch 94/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9606 -
accuracy: 0.6855 - val_loss: 0.9283 - val_accuracy: 0.6903 - lr: 1.0000e-13
Epoch 95/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9644 -
accuracy: 0.6882 - val_loss: 0.9283 - val_accuracy: 0.6904 - lr: 1.0000e-13
Epoch 96/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9600 -
accuracy: 0.6866 - val_loss: 0.9273 - val_accuracy: 0.6916 - lr: 1.0000e-13
Epoch 97/100
600/600 [==============================] - 5s 8ms/step - loss: 0.9608 -
accuracy: 0.6883 - val_loss: 0.9283 - val_accuracy: 0.6911 - lr: 1.0000e-13
Epoch 98/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9610 -
accuracy: 0.6893 - val_loss: 0.9283 - val_accuracy: 0.6911 - lr: 1.0000e-14
Epoch 99/100
600/600 [==============================] - 6s 10ms/step - loss: 0.9551 -
accuracy: 0.6873 - val_loss: 0.9280 - val_accuracy: 0.6909 - lr: 1.0000e-14
Epoch 100/100
600/600 [==============================] - 5s 9ms/step - loss: 0.9586 -
accuracy: 0.6877 - val_loss: 0.9293 - val_accuracy: 0.6917 - lr: 1.0000e-14
```




```
375/375 [==============================] - 2s 4ms/step - loss: 0.9108 -
accuracy: 0.6957
Test accuracy: 0.696
```

[ ]:
```
final_model.save("jain_ayushri_csce636_project_1" + ".h5")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

[ ]:
```
from google.colab import files
files.download('jain_ayushri_csce636_project_1.h5')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

4 Final Cell To Be Run For Grading

[1]:
```
import warnings
warnings.filterwarnings('ignore')
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
os.environ["CUDA_VISIBLE_DEVICES"] = "0"


# print(f"Your accuracy on Train Set: {test_acc}")
```

[4]:
```
from tensorflow.keras import models
import pickle
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
import numpy as np
from tensorflow.python.ops.numpy_ops import np_config
np_config.enable_numpy_behavior()

model = models.load_model("./jain_ayushri_csce636_project_1.h5")
test_labels = pickle.load(open("./636_project1_train_labels", 'rb'))
test_images = pickle.load(open("./636_project1_train_images", 'rb'))

# Include your data preprocessing code if applicable
# reshape
test_images = test_images.reshape((test_images.shape[0], 28, 28, 1))
test_images = test_images.astype('float') / 255.

# add padding
test_images = np.array(test_images)
test_images = np.pad(test_images, ((0,0),(2,2),(2,2),(0,0)), 'constant')

# standardization
mean_px = test_images.mean().astype(np.float32)
std_px = test_images.std().astype(np.float32)
test_images = (test_images - mean_px)/(std_px)

# one-hot encoding the labels
test_labels = to_categorical(test_labels, num_classes = 10)
```

```
# Include your data preprocessing code if applicable

test_loss, test_acc = model.evaluate(test_images, test_labels)
your_score = round(test_acc*1000) / 10
print(f"Your accuracy on Train Set: {test_acc}")
```

```
1875/1875 [==============================] - 12s 6ms/step - loss: 0.7754 -
accuracy: 0.7402
Your accuracy on Train Set: 0.7401999831199646
```

[3]:
```
from tensorflow.keras import models
import pickle
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
import numpy as np
from tensorflow.python.ops.numpy_ops import np_config
np_config.enable_numpy_behavior()

model = models.load_model("./jain_ayushri_csce636_project_1.h5")
test_labels = pickle.load(open("./636_project1_test_labels", 'rb'))
test_images = pickle.load(open("./636_project1_test_images", 'rb'))

# Include your data preprocessing code if applicable
# reshape
test_images = test_images.reshape((test_images.shape[0], 28, 28, 1))
test_images = test_images.astype('float') / 255.

# add padding
test_images = np.array(test_images)
test_images = np.pad(test_images, ((0,0),(2,2),(2,2),(0,0)), 'constant')

# standardization
mean_px = test_images.mean().astype(np.float32)
std_px = test_images.std().astype(np.float32)
test_images = (test_images - mean_px)/(std_px)

# one-hot encoding the labels
test_labels = to_categorical(test_labels, num_classes = 10)
# Include your data preprocessing code if applicable

test_loss, test_acc = model.evaluate(test_images, test_labels)
your_score = round(test_acc*1000) / 10
print(f"Your Score: {your_score}")
```

```
313/313 [==============================] - 5s 7ms/step - loss: 0.9116 -
accuracy: 0.6971
Your Score: 69.7
```

```
[ ]:
```