Updates and Imports

```
!sudo apt update
!sudo apt autoremove
!sudo apt upgrade
!sudo apt install python3x
```

```
!python --version
```

```
import sys
print("version:", sys.version)
```

```
!pip install -U pandas
```

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pandas in
/usr/local/lib/python3.8/dist-packages (1.3.5)
Collecting pandas
  Downloading pandas-1.5.2-cp38-cp38-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.2 MB)
ent already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.8/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: numpy>=1.20.3 in
/usr/local/lib/python3.8/dist-packages (from pandas) (1.21.6)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.8/dist-packages (from pandas) (2022.6)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.8.1-
>pandas) (1.15.0)
Installing collected packages: pandas
  Attempting uninstall: pandas
    Found existing installation: pandas 1.3.5
    Uninstalling pandas-1.3.5:
      Successfully uninstalled pandas-1.3.5
Successfully installed pandas-1.5.2
```

{"pip_warning":{"packages":["pandas"]}}

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly
remount, call drive.mount("/content/drive/", force_remount=True).
```

```
import seaborn as sns
import cv2
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
        BatchNormalization
from tensorflow.keras.layers import Dense, Dropout, Flatten,
        Activation
from tensorflow.keras.metrics import categorical_accuracy,
        top_k_categorical_accuracy, categorical_crossentropy
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping,
        ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import regularizers
from tensorflow.keras.applications import MobileNet,MobileNetV2
from tensorflow.keras.applications.mobilenet import preprocess_input
import matplotlib.pyplot as plt
from tensorflow.keras.utils import to_categorical
```

```
import pandas as pd
import numpy as np
from numpy import newaxis
from sklearn.model_selection import train_test_split
import pickle
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
import cv2
```

```
from numpy import newaxis
from sklearn.model_selection import train_test_split
import pickle
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
import cv2
```

```
train_path =
        "/content/drive/MyDrive/AMLminiproject/train100c5k_v2.pkl"
test_path =
        "/content/drive/MyDrive/AMLminiproject/test100c5k_nolabel.pkl"
# train_data = open(train_path,'rb')
# test_data = open(test_path,'rb')

# df_train = pd.read_pickle(train_path)
# df_test = pd.read_pickle(test_path)
df_train = pickle.load(open(train_path,'rb'))
df_test = pickle.load(open(test_path,'rb'))
# train_data = df['data'].values
```

Mobilenet

reference: https://www.kaggle.com/code/sabarish2611/alexnet-vs-mobilenet-using-mnist-data#AlexNet-vs-MobileNet-:-Comparing-the-performance

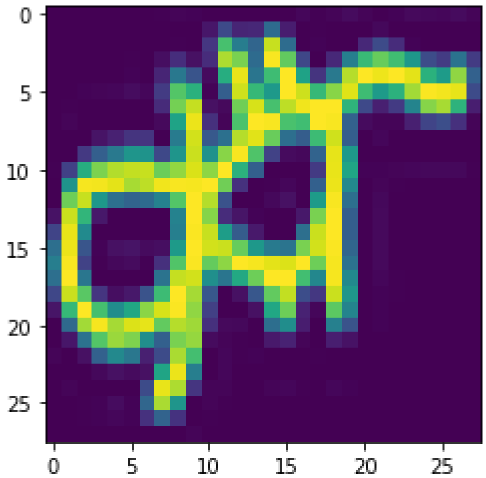
```
# sampling
sample = df_train.groupby('target', group_keys=False).apply(lambda x:
        x.sample(frac=0.3))
sample['data'] = sample['data']/255

X = np.array(list(sample['data'].values))
# X = np.array([img[:,:,newaxis] for img in
        np.array(sample['data'].values)])
y = np.array(sample['target'].values)
df_train = None
df_test = None
sample = None
X.shape
```

```
(150000, 28, 28)
```

```
plt.imshow(X[0])
```

```
<matplotlib.image.AxesImage at 0x7f8cd35c4d30>
```




```
X_train, X_test, y_train, y_test = train_test_split(X, y,
        random_state=42)
```

```
X = None
```

```
y = None
```

```
X_train.shape
```

```
(112500, 28, 28)
```

```
# X_re = np.zeros((X.shape[0],32,32,3))
# for i in tqdm(range(X.shape[0])):
#     temp = cv2.resize(X[i], dsize=(32, 32),
        interpolation=cv2.INTER_CUBIC)
#     # temp = temp[:,:,newaxis]
#     # print(temp.shape)
#     img = np.zeros((32,32,3))

#     # print(img.shape)
#     img[:,:,0] = temp
#     img[:,:,1] = temp
#     img[:,:,2] = temp
#     X_re[i] = img
X_train_re = np.zeros((X_train.shape[0],32,32,3))
for i in tqdm(range(X_train.shape[0])):
    temp = cv2.resize(X_train[i], dsize=(32, 32),
        interpolation=cv2.INTER_CUBIC)
    # temp = temp[:,:,newaxis]
    # print(temp.shape)
    img = np.zeros((32,32,3))

    # print(img.shape)
    img[:,:,0] = temp
```

```
    img[:,:,1] = temp
    img[:,:,2] = temp
    X_train_re[i] = img
    X_train[i] = None
X_train = None
X_test_re = np.zeros((X_test.shape[0],32,32,3))
for i in tqdm(range(X_test.shape[0])):
    temp = cv2.resize(X_test[i], dsize=(32, 32),
        interpolation=cv2.INTER_CUBIC)
    # temp = temp[:,:,newaxis]
    # print(temp.shape)
    img = np.zeros((32,32,3))

    # print(img.shape)
    img[:,:,0] = temp
    img[:,:,1] = temp
    img[:,:,2] = temp
    X_test_re[i] = img
    X_test[i] = None

X_test = None
```

{"version_major":2,"version_minor":0,"model_id":"5129e5f8573d4a298f261c9ecc7e3ea7"}

{"version_major":2,"version_minor":0,"model_id":"7a4f5d610cf3456fbbf2914b758803cb"}

```
X_train_re.shape
```

```
(112500, 32, 32, 3)
```

```
base_model = MobileNet(include_top=False,
        weights='imagenet',input_shape = (32,32,3), classes=100)
base_model.trainable = True


for layer in base_model.layers[:50]:
    layer.trainable =  False


MobileNet_model = Sequential()
MobileNet_model.add(base_model)
MobileNet_model.add(Flatten())
MobileNet_model.add(Dense(100,activation=('softmax')))
```

```
early_stopping = EarlyStopping(min_delta = 0.001,patience =
        20,restore_best_weights = True,verbose = 0)


# Compile
MobileNet_model.compile(optimizer = "adam" , loss =
        'sparse_categorical_crossentropy' , metrics = ['accuracy'])


# Train
Mobile = MobileNet_model.fit(X_train_re, y_train, batch_size = 400,
        epochs = 50,callbacks = [early_stopping], validation_data =
        (X_test_re, y_test))


MobileNet_model.summary()
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows`
is not in [128, 160, 192, 224]. Weights for input shape (224, 224)
will be loaded as the default.

Epoch 1/50
282/282 [==============================] - 22s 34ms/step - loss:
2.3421 - accuracy: 0.4379 - val_loss: 2.2859 - val_accuracy: 0.4241
Epoch 2/50
282/282 [==============================] - 7s 26ms/step - loss: 1.5940
- accuracy: 0.5846 - val_loss: 1.7769 - val_accuracy: 0.5469
Epoch 3/50
282/282 [==============================] - 8s 28ms/step - loss: 1.3610
- accuracy: 0.6362 - val_loss: 1.7431 - val_accuracy: 0.5750
Epoch 4/50
282/282 [==============================] - 7s 27ms/step - loss: 1.2019
- accuracy: 0.6715 - val_loss: 1.6661 - val_accuracy: 0.5821
Epoch 5/50
282/282 [==============================] - 7s 26ms/step - loss: 1.0716
- accuracy: 0.7021 - val_loss: 1.7595 - val_accuracy: 0.5697
Epoch 6/50
282/282 [==============================] - 7s 26ms/step - loss: 0.9593
- accuracy: 0.7280 - val_loss: 1.7388 - val_accuracy: 0.5848
Epoch 7/50
282/282 [==============================] - 7s 26ms/step - loss: 0.8498
- accuracy: 0.7543 - val_loss: 1.8712 - val_accuracy: 0.5755
Epoch 8/50
282/282 [==============================] - 7s 26ms/step - loss: 0.7601
- accuracy: 0.7752 - val_loss: 1.8884 - val_accuracy: 0.5788
Epoch 9/50
```

```
282/282 [==============================] - 8s 29ms/step - loss: 0.6592
- accuracy: 0.8015 - val_loss: 2.0165 - val_accuracy: 0.5817
Epoch 10/50
282/282 [==============================] - 9s 32ms/step - loss: 0.5815
- accuracy: 0.8218 - val_loss: 2.1094 - val_accuracy: 0.5749
Epoch 11/50
282/282 [==============================] - 9s 32ms/step - loss: 0.5015
- accuracy: 0.8446 - val_loss: 2.2363 - val_accuracy: 0.5777
Epoch 12/50
282/282 [==============================] - 8s 29ms/step - loss: 0.4281
- accuracy: 0.8653 - val_loss: 2.2120 - val_accuracy: 0.5758
Epoch 13/50
282/282 [==============================] - 7s 26ms/step - loss: 0.3791
- accuracy: 0.8791 - val_loss: 2.3544 - val_accuracy: 0.5704
Epoch 14/50
282/282 [==============================] - 7s 26ms/step - loss: 0.3340
- accuracy: 0.8926 - val_loss: 2.3482 - val_accuracy: 0.5773
Epoch 15/50
282/282 [==============================] - 7s 26ms/step - loss: 0.2903
- accuracy: 0.9072 - val_loss: 2.4643 - val_accuracy: 0.5824
Epoch 16/50
282/282 [==============================] - 8s 27ms/step - loss: 0.2675
- accuracy: 0.9130 - val_loss: 2.4808 - val_accuracy: 0.5766
Epoch 17/50
282/282 [==============================] - 7s 26ms/step - loss: 0.2334
- accuracy: 0.9242 - val_loss: 2.5780 - val_accuracy: 0.5808
Epoch 18/50
282/282 [==============================] - 8s 28ms/step - loss: 0.2231
- accuracy: 0.9278 - val_loss: 2.6697 - val_accuracy: 0.5767
Epoch 19/50
282/282 [==============================] - 7s 26ms/step - loss: 0.2066
- accuracy: 0.9327 - val_loss: 2.6816 - val_accuracy: 0.5785
Epoch 20/50
282/282 [==============================] - 7s 26ms/step - loss: 0.1954
- accuracy: 0.9373 - val_loss: 2.7885 - val_accuracy: 0.5827
Epoch 21/50
282/282 [==============================] - 7s 26ms/step - loss: 0.1681
- accuracy: 0.9448 - val_loss: 2.7846 - val_accuracy: 0.5888
Epoch 22/50
282/282 [==============================] - 7s 26ms/step - loss: 0.1701
```

```
- accuracy: 0.9436 - val_loss: 2.7298 - val_accuracy: 0.5878
Epoch 23/50
282/282 [==============================] - 7s 26ms/step - loss: 0.1674
- accuracy: 0.9451 - val_loss: 2.8234 - val_accuracy: 0.5800
Epoch 24/50
282/282 [==============================] - 8s 27ms/step - loss: 0.1504
- accuracy: 0.9499 - val_loss: 2.8720 - val_accuracy: 0.5853
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 mobilenet_1.00_224 (Functio  (None, 1, 1, 1024)       3228864   
 nal)                                                            
                                                                 
 flatten (Flatten)           (None, 1024)              0         
                                                                 
 dense (Dense)               (None, 100)               102500    
                                                                 
=================================================================
Total params: 3,331,364
Trainable params: 2,766,948
Non-trainable params: 564,416
_________________________________________________________________
```

```python
base_model = MobileNet(include_top=False,
        weights='imagenet',input_shape = (32,32,3), classes=100)
base_model.trainable = True

for layer in base_model.layers[:50]:
    layer.trainable =  False

MobileNet_model = Sequential()
MobileNet_model.add(base_model)
MobileNet_model.add(Flatten())
MobileNet_model.add(Dense(100,activation=('softmax')))

early_stopping = EarlyStopping(min_delta = 0.001,patience =
        20,restore_best_weights = True,verbose = 0)

# Compile
MobileNet_model.compile(optimizer = "adam" , loss =
        'sparse_categorical_crossentropy' , metrics = ['accuracy'])
```

```
# Train
Mobile = MobileNet_model.fit(X_train_re, y_train, batch_size = 400,
        epochs = 50,callbacks = [early_stopping], validation_data =
        (X_test_re, y_test))
```

```
MobileNet_model.summary()

MobileNet_model.save('/content/drive/MyDrive/AML mini
        project/mobile.h5')

# early_stopping = EarlyStopping(min_delta = 0.001,patience =
        20,restore_best_weights = True,verbose = 2)
# model = MobileNet(input_shape=(32, 32, 3), alpha=1., weights=None,
        classes=100)
# model.compile(optimizer=Adam(lr=0.002),
        loss='categorical_crossentropy',
#              metrics=['accuracy'])


# Mobile = model.fit(X_train_re, y_train, batch_size = 400, epochs =
        50, callbacks = [early_stopping], validation_data =
        (X_test_re, y_test))
# print(model.summary())
```

LeNet

reference: https://towardsdatascience.com/going-beyond-99-mnist-handwritten-digits-recognition-cfff96337392

```
train_path =
        "/content/drive/MyDrive/AMLminiproject/train100c5k_v2.pkl"
test_path =
        "/content/drive/MyDrive/AMLminiproject/test100c5k_nolabel.pkl"
# train_data = open(train_path,'rb')
# test_data = open(test_path,'rb')

# df_train = pd.read_pickle(train_path)
# df_test = pd.read_pickle(test_path)
df_train = pickle.load(open(train_path,'rb'))
df_test = pickle.load(open(test_path,'rb'))
# train_data = df['data'].values

# sampling
#sample = df_train.groupby('target', group_keys=False).apply(lambda x:
        x.sample(frac=0.3))
sample = df_train.copy()
sample['data'] = sample['data']/255
```

```
# X = np.array(list(sample['data'].values))
X = np.array([img[:,:,newaxis] for img in
        np.array(sample['data'].values)])
y = np.array(sample['target'].values)
df_train = None
df_test = None
sample = None
X.shape
```

```
(500000, 28, 28, 1)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
        random_state=42)
X = None
y = None
X_train.shape
```

```
(375000, 28, 28, 1)
```

```
X_train.shape
```

```
(375000, 28, 28, 1)
```

```
X_train_re = np.pad(X_train, ((0,0),(2,2),(2,2),(0,0)), 'constant')
X_test_re = np.pad(X_test, ((0,0),(2,2),(2,2),(0,0)), 'constant')
X_train = None
X_test = None
X_train_re.shape
```

```
(375000, 32, 32, 1)
```

```
y_train = to_categorical(y_train, 100)
y_test = to_categorical(y_test, 100)
```

LeNet architecture with additional hidden layers and batch normalization followed by them

```
model = Sequential([
# Layer 1
Conv2D(filters = 32, kernel_size = 5, strides = 1, activation =
        'relu', input_shape = (32,32,1),
        kernel_regularizer=regularizers.l1_l2(l1=0, l2=0.0005)),
# Layer 2
Conv2D(filters = 32, kernel_size = 5, strides = 1, use_bias=False),
# Layer 3
```

```
        BatchNormalization(),
        # ------------------- #
        Activation('relu'),
        MaxPooling2D(pool_size = 2, strides = 2),
        Dropout(0.25),
        # ------------------- #
        # Layer 3
        Conv2D(filters = 64, kernel_size = 3, strides = 1, activation =
                'relu', kernel_regularizer=regularizers.l1_l2(l1=0,
                l2=0.0005)),
        # Layer 4
        Conv2D(filters = 64, kernel_size = 3, strides = 1, use_bias=False),
        # Layer 5
        BatchNormalization(),
        # ------------------- #
        Activation('relu'),
        MaxPooling2D(pool_size = 2, strides = 2),
        Dropout(0.25),
        Flatten(),
        # ------------------- #
        # Layer 6
        Dense(units = 256, use_bias=False),
        # Layer 7
        BatchNormalization(),
        # ------------------- #
        Activation('relu'),
        # ------------------- #
        # Layer 8
        Dense(units = 128, use_bias=False),
        # Layer 9
        BatchNormalization(),
        # ------------------- #
        Activation('relu'),
        # ------------------- #
        # Layer 10
        Dense(units = 84, use_bias=False),
        # Layer 11
        BatchNormalization(),
        # ------------------- #
        Activation('relu'),
        Dropout(0.25),
```

```
# - - - - - - - - - - - - - - - - #
# Output
Dense(units = 100, activation = 'softmax')
])

early_stopping = EarlyStopping(min_delta = 0.001,patience =
        20,restore_best_weights = True,verbose = 0)
model.compile(optimizer='adam', loss='categorical_crossentropy',
        metrics=['accuracy'])

# model.fit(X_train_re, y_train, batch_size = 400, epochs =
        50,callbacks = [early_stopping], validation_data = (X_test_re,
        y_test))
history = model.fit(X_train_re, y_train, epochs = 50, batch_size =
        100, callbacks = [early_stopping], validation_data =
        (X_test_re, y_test))
```

```
Epoch 1/50
3750/3750 [==============================] - 54s 11ms/step - loss:
2.2783 - accuracy: 0.4449 - val_loss: 1.6516 - val_accuracy: 0.5801
Epoch 2/50
3750/3750 [==============================] - 42s 11ms/step - loss:
1.6669 - accuracy: 0.5811 - val_loss: 1.4732 - val_accuracy: 0.6205
Epoch 3/50
3750/3750 [==============================] - 35s 9ms/step - loss:
1.5325 - accuracy: 0.6130 - val_loss: 1.3022 - val_accuracy: 0.6662
Epoch 4/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.4575 - accuracy: 0.6315 - val_loss: 1.2603 - val_accuracy: 0.6777
Epoch 5/50
3750/3750 [==============================] - 35s 9ms/step - loss:
1.4107 - accuracy: 0.6435 - val_loss: 1.2330 - val_accuracy: 0.6849
Epoch 6/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.3774 - accuracy: 0.6511 - val_loss: 1.2859 - val_accuracy: 0.6710
Epoch 7/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.3474 - accuracy: 0.6589 - val_loss: 1.1728 - val_accuracy: 0.6997
Epoch 8/50
3750/3750 [==============================] - 35s 9ms/step - loss:
1.3262 - accuracy: 0.6645 - val_loss: 1.1501 - val_accuracy: 0.7063
Epoch 9/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.3089 - accuracy: 0.6690 - val_loss: 1.2101 - val_accuracy: 0.6926
```

```
Epoch 10/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.2971 - accuracy: 0.6720 - val_loss: 1.2375 - val_accuracy: 0.6868
Epoch 11/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.2832 - accuracy: 0.6750 - val_loss: 1.1592 - val_accuracy: 0.7046
Epoch 12/50
3750/3750 [==============================] - 35s 9ms/step - loss:
1.2681 - accuracy: 0.6789 - val_loss: 1.1577 - val_accuracy: 0.7052
Epoch 13/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.2565 - accuracy: 0.6821 - val_loss: 1.1342 - val_accuracy: 0.7117
Epoch 14/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.2473 - accuracy: 0.6844 - val_loss: 1.1447 - val_accuracy: 0.7090
Epoch 15/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.2405 - accuracy: 0.6861 - val_loss: 1.1253 - val_accuracy: 0.7134
Epoch 16/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.2326 - accuracy: 0.6877 - val_loss: 1.1250 - val_accuracy: 0.7139
Epoch 17/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.2237 - accuracy: 0.6898 - val_loss: 1.0995 - val_accuracy: 0.7190
Epoch 18/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.2178 - accuracy: 0.6915 - val_loss: 1.1277 - val_accuracy: 0.7145
Epoch 19/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.2120 - accuracy: 0.6925 - val_loss: 1.0959 - val_accuracy: 0.7211
Epoch 20/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.2063 - accuracy: 0.6940 - val_loss: 1.0966 - val_accuracy: 0.7214
Epoch 21/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.2006 - accuracy: 0.6961 - val_loss: 1.1016 - val_accuracy: 0.7194
Epoch 22/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1943 - accuracy: 0.6963 - val_loss: 1.1236 - val_accuracy: 0.7140
Epoch 23/50
```

```
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1898 - accuracy: 0.6979 - val_loss: 1.1178 - val_accuracy: 0.7157
Epoch 24/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1842 - accuracy: 0.6994 - val_loss: 1.0987 - val_accuracy: 0.7211
Epoch 25/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.1826 - accuracy: 0.6997 - val_loss: 1.0938 - val_accuracy: 0.7225
Epoch 26/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1776 - accuracy: 0.7012 - val_loss: 1.0939 - val_accuracy: 0.7221
Epoch 27/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.1742 - accuracy: 0.7020 - val_loss: 1.0851 - val_accuracy: 0.7240
Epoch 28/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.1715 - accuracy: 0.7027 - val_loss: 1.0933 - val_accuracy: 0.7220
Epoch 29/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1690 - accuracy: 0.7031 - val_loss: 1.0715 - val_accuracy: 0.7274
Epoch 30/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.1622 - accuracy: 0.7045 - val_loss: 1.0896 - val_accuracy: 0.7241
Epoch 31/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1633 - accuracy: 0.7047 - val_loss: 1.0927 - val_accuracy: 0.7245
Epoch 32/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1583 - accuracy: 0.7061 - val_loss: 1.0938 - val_accuracy: 0.7230
Epoch 33/50
3750/3750 [==============================] - 32s 8ms/step - loss:
1.1554 - accuracy: 0.7065 - val_loss: 1.0898 - val_accuracy: 0.7233
Epoch 34/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1521 - accuracy: 0.7068 - val_loss: 1.0800 - val_accuracy: 0.7269
Epoch 35/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1514 - accuracy: 0.7075 - val_loss: 1.0962 - val_accuracy: 0.7238
Epoch 36/50
3750/3750 [==============================] - 34s 9ms/step - loss:
```

```
1.1505 - accuracy: 0.7074 - val_loss: 1.1167 - val_accuracy: 0.7165
Epoch 37/50
3750/3750 [==============================] - 32s 8ms/step - loss:
1.1466 - accuracy: 0.7085 - val_loss: 1.0998 - val_accuracy: 0.7219
Epoch 38/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1423 - accuracy: 0.7099 - val_loss: 1.0994 - val_accuracy: 0.7205
Epoch 39/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1412 - accuracy: 0.7095 - val_loss: 1.0968 - val_accuracy: 0.7220
Epoch 40/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1361 - accuracy: 0.7107 - val_loss: 1.0621 - val_accuracy: 0.7308
Epoch 41/50
3750/3750 [==============================] - 32s 8ms/step - loss:
1.1384 - accuracy: 0.7110 - val_loss: 1.0644 - val_accuracy: 0.7309
Epoch 42/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1306 - accuracy: 0.7119 - val_loss: 1.0785 - val_accuracy: 0.7276
Epoch 43/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1335 - accuracy: 0.7110 - val_loss: 1.0780 - val_accuracy: 0.7275
Epoch 44/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1282 - accuracy: 0.7127 - val_loss: 1.1323 - val_accuracy: 0.7139
Epoch 45/50
3750/3750 [==============================] - 35s 9ms/step - loss:
1.1290 - accuracy: 0.7122 - val_loss: 1.0585 - val_accuracy: 0.7326
Epoch 46/50
3750/3750 [==============================] - 32s 8ms/step - loss:
1.1273 - accuracy: 0.7128 - val_loss: 1.0745 - val_accuracy: 0.7280
Epoch 47/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.1273 - accuracy: 0.7129 - val_loss: 1.0550 - val_accuracy: 0.7332
Epoch 48/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1233 - accuracy: 0.7137 - val_loss: 1.0631 - val_accuracy: 0.7302
Epoch 49/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1224 - accuracy: 0.7137 - val_loss: 1.0843 - val_accuracy: 0.7255
```

```
Epoch 50/50
3750/3750 [==============================] - 40s 11ms/step - loss:
1.1189 - accuracy: 0.7147 - val_loss: 1.0673 - val_accuracy: 0.7298
```

```
model.save('/content/drive/MyDrive/AML mini project/lenet.h5')
```

```
save_path = '/content/drive/MyDrive/AML mini project/lenet.h5'
model.save(save_path,save_format='tf')
```

```
save_path = '/content/drive/MyDrive/AMLminiproject/lenet.h5'
model = tf.keras.models.load_model(save_path)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        832       
                                                                 
 conv2d_1 (Conv2D)           (None, 24, 24, 32)        25600     
                                                                 
 batch_normalization (BatchN  (None, 24, 24, 32)       128       
 ormalization)                                                   
                                                                 
 activation (Activation)     (None, 24, 24, 32)        0         
                                                                 
 max_pooling2d (MaxPooling2D  (None, 12, 12, 32)       0         
 )                                                               
                                                                 
 dropout (Dropout)           (None, 12, 12, 32)        0         
                                                                 
 conv2d_2 (Conv2D)           (None, 10, 10, 64)        18496     
                                                                 
 conv2d_3 (Conv2D)           (None, 8, 8, 64)          36864     
                                                                 
 batch_normalization_1 (Batc  (None, 8, 8, 64)         256       
 hNormalization)                                                 
                                                                 
 activation_1 (Activation)   (None, 8, 8, 64)          0         
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 4, 4, 64)         0         
 2D)                                                             
```

```
 dropout_1 (Dropout)         (None, 4, 4, 64)          0

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 256)               262144

 batch_normalization_2 (Batc  (None, 256)              1024
 hNormalization)

 activation_2 (Activation)   (None, 256)               0

 dense_1 (Dense)             (None, 128)               32768

 batch_normalization_3 (Batc  (None, 128)              512
 hNormalization)

 activation_3 (Activation)   (None, 128)               0

 dense_2 (Dense)             (None, 84)                10752

 batch_normalization_4 (Batc  (None, 84)               336
 hNormalization)

 activation_4 (Activation)   (None, 84)                0

 dropout_2 (Dropout)         (None, 84)                0

 dense_3 (Dense)             (None, 100)               8500

=================================================================
Total params: 398,212
Trainable params: 397,084
Non-trainable params: 1,128
_________________________________________________________________
```

```
# training for more 50 epochs i.e total of 100 epochs
early_stopping = EarlyStopping(min_delta = 0.001,patience =
        20,restore_best_weights = True,verbose = 0)
```

```
history = model.fit(X_train_re, y_train, epochs = 50, batch_size =
        100, callbacks = [early_stopping], validation_data =
        (X_test_re, y_test))
```

```
Epoch 1/50
3750/3750 [==============================] - 47s 10ms/step - loss:
1.1188 - accuracy: 0.7156 - val_loss: 1.0998 - val_accuracy: 0.7250
Epoch 2/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.1188 - accuracy: 0.7154 - val_loss: 1.0536 - val_accuracy: 0.7333
Epoch 3/50
3750/3750 [==============================] - 36s 10ms/step - loss:
1.1176 - accuracy: 0.7152 - val_loss: 1.0578 - val_accuracy: 0.7320
Epoch 4/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1138 - accuracy: 0.7159 - val_loss: 1.0859 - val_accuracy: 0.7266
Epoch 5/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.1124 - accuracy: 0.7159 - val_loss: 1.0568 - val_accuracy: 0.7334
Epoch 6/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1119 - accuracy: 0.7168 - val_loss: 1.0853 - val_accuracy: 0.7261
Epoch 7/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1092 - accuracy: 0.7173 - val_loss: 1.0525 - val_accuracy: 0.7336
Epoch 8/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1099 - accuracy: 0.7166 - val_loss: 1.0811 - val_accuracy: 0.7275
Epoch 9/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1091 - accuracy: 0.7174 - val_loss: 1.0553 - val_accuracy: 0.7328
Epoch 10/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1065 - accuracy: 0.7175 - val_loss: 1.0673 - val_accuracy: 0.7310
Epoch 11/50
3750/3750 [==============================] - 35s 9ms/step - loss:
1.1051 - accuracy: 0.7182 - val_loss: 1.0560 - val_accuracy: 0.7336
Epoch 12/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.1051 - accuracy: 0.7188 - val_loss: 1.0638 - val_accuracy: 0.7315
```

```
Epoch 13/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.1011 - accuracy: 0.7195 - val_loss: 1.0767 - val_accuracy: 0.7279
Epoch 14/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1019 - accuracy: 0.7186 - val_loss: 1.0484 - val_accuracy: 0.7347
Epoch 15/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.1014 - accuracy: 0.7193 - val_loss: 1.0647 - val_accuracy: 0.7318
Epoch 16/50
3750/3750 [==============================] - 35s 9ms/step - loss:
1.0990 - accuracy: 0.7199 - val_loss: 1.0630 - val_accuracy: 0.7326
Epoch 17/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.0970 - accuracy: 0.7200 - val_loss: 1.0759 - val_accuracy: 0.7294
Epoch 18/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.0971 - accuracy: 0.7206 - val_loss: 1.0858 - val_accuracy: 0.7267
Epoch 19/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.0967 - accuracy: 0.7205 - val_loss: 1.0699 - val_accuracy: 0.7298
Epoch 20/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.0963 - accuracy: 0.7207 - val_loss: 1.0471 - val_accuracy: 0.7356
Epoch 21/50
3750/3750 [==============================] - 35s 9ms/step - loss:
1.0961 - accuracy: 0.7209 - val_loss: 1.0690 - val_accuracy: 0.7310
Epoch 22/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.0943 - accuracy: 0.7215 - val_loss: 1.0556 - val_accuracy: 0.7338
Epoch 23/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.0939 - accuracy: 0.7212 - val_loss: 1.0651 - val_accuracy: 0.7323
Epoch 24/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.0921 - accuracy: 0.7212 - val_loss: 1.0574 - val_accuracy: 0.7337
Epoch 25/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.0924 - accuracy: 0.7207 - val_loss: 1.0836 - val_accuracy: 0.7282
Epoch 26/50
```

```
3750/3750 [==============================] - 35s 9ms/step - loss:
1.0929 - accuracy: 0.7214 - val_loss: 1.0697 - val_accuracy: 0.7306
Epoch 27/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.0901 - accuracy: 0.7218 - val_loss: 1.0572 - val_accuracy: 0.7345
Epoch 28/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.0885 - accuracy: 0.7227 - val_loss: 1.0650 - val_accuracy: 0.7323
Epoch 29/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.0878 - accuracy: 0.7228 - val_loss: 1.0554 - val_accuracy: 0.7341
Epoch 30/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.0869 - accuracy: 0.7233 - val_loss: 1.0543 - val_accuracy: 0.7348
Epoch 31/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.0873 - accuracy: 0.7232 - val_loss: 1.0550 - val_accuracy: 0.7342
Epoch 32/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.0871 - accuracy: 0.7227 - val_loss: 1.0521 - val_accuracy: 0.7358
Epoch 33/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.0861 - accuracy: 0.7234 - val_loss: 1.0510 - val_accuracy: 0.7351
Epoch 34/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.0860 - accuracy: 0.7228 - val_loss: 1.0557 - val_accuracy: 0.7346
Epoch 35/50
3750/3750 [==============================] - 35s 9ms/step - loss:
1.0829 - accuracy: 0.7234 - val_loss: 1.0761 - val_accuracy: 0.7296
Epoch 36/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.0814 - accuracy: 0.7245 - val_loss: 1.0670 - val_accuracy: 0.7321
Epoch 37/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.0833 - accuracy: 0.7238 - val_loss: 1.0397 - val_accuracy: 0.7382
Epoch 38/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.0814 - accuracy: 0.7232 - val_loss: 1.0523 - val_accuracy: 0.7361
Epoch 39/50
3750/3750 [==============================] - 34s 9ms/step - loss:
```

```
1.0819 - accuracy: 0.7242 - val_loss: 1.0573 - val_accuracy: 0.7328
Epoch 40/50
3750/3750 [==============================] - 35s 9ms/step - loss:
1.0786 - accuracy: 0.7248 - val_loss: 1.0673 - val_accuracy: 0.7314
Epoch 41/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.0809 - accuracy: 0.7241 - val_loss: 1.0668 - val_accuracy: 0.7325
Epoch 42/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.0780 - accuracy: 0.7249 - val_loss: 1.0547 - val_accuracy: 0.7353
Epoch 43/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.0772 - accuracy: 0.7252 - val_loss: 1.0579 - val_accuracy: 0.7352
Epoch 44/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.0773 - accuracy: 0.7251 - val_loss: 1.0533 - val_accuracy: 0.7353
Epoch 45/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.0762 - accuracy: 0.7252 - val_loss: 1.0487 - val_accuracy: 0.7371
Epoch 46/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.0765 - accuracy: 0.7251 - val_loss: 1.0524 - val_accuracy: 0.7354
Epoch 47/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.0748 - accuracy: 0.7252 - val_loss: 1.0475 - val_accuracy: 0.7381
Epoch 48/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.0743 - accuracy: 0.7253 - val_loss: 1.0653 - val_accuracy: 0.7329
Epoch 49/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.0733 - accuracy: 0.7259 - val_loss: 1.0532 - val_accuracy: 0.7353
Epoch 50/50
3750/3750 [==============================] - 35s 9ms/step - loss:
1.0739 - accuracy: 0.7252 - val_loss: 1.0760 - val_accuracy: 0.7325
```

```
save_path = '/content/drive/MyDrive/AML mini project/lenet_100e.h5'
model.save(save_path,save_format='tf')
```

```
save_path = '/content/drive/MyDrive/AML mini project/lenet.h5'
model = tf.keras.models.load_model(save_path)
```

```
# we can see that the best fit is around 75 epochs, hence we will
        train the 50 epoch model for mre 25 epochs total of 75
history = model.fit(X_train_re, y_train, epochs = 25, batch_size =
        100, validation_data = (X_test_re, y_test))
```

```
----------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-17-2bf365c77103> in <module>
      1 # we can see that the best fit is around 75 epochs, hence we
will train the 50 epoch model for mre 25 epochs total of 75
----> 2 history = model.fit(X_train_re, y_train, epochs = 25,
batch_size = 100, validation_data = (X_test_re, y_test))

NameError: name 'model' is not defined
```

```
save_path = '/content/drive/MyDrive/AML mini project/lenet_75e.h5'
model.save(save_path,save_format='tf')
```

```
# df_test['data'] = df_test['data']/255
df_test.head()
```

	data
0	[[0, 0, 0, 4, 4, 0, 0, 3, 5, 2, 0, 0, 11, 0, 0...
1	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
2	[[0, 3, 0, 0, 101, 128, 21, 0, 10, 4, 3, 2, 2,...
3	[[0, 12, 55, 101, 116, 98, 79, 66, 52, 39, 30,...
4	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...

```
df_test['data'] = df_test['data']/255
```

```
df_test.head()
```

```
----------------------------------------------------------------------
-----
AttributeError                            Traceback (most recent call
last)
<ipython-input-27-711c74f5b217> in <module>
----> 1 df_test.head()
```

```
AttributeError: 'NoneType' object has no attribute 'head'
```

```
test_data = np.array([img[:,:,newaxis] for img in
        np.array(df_test['data'].values)])
test_data.shape
```

```
(100000, 28, 28, 1)
```

```
X_test_data = np.pad(test_data, ((0,0),(2,2),(2,2),(0,0)), 'constant')
X_test_data.shape
```

```
(100000, 32, 32, 1)
```

```
save_path = '/content/drive/MyDrive/AML mini project/lenet_75e.h5'
model = tf.keras.models.load_model(save_path)
```

```
X_test_data[0].shape
```

```
(32, 32, 1)
```

```
y_pred = model.predict(X_test_data)
```

```
3125/3125 [==============================] - 7s 2ms/step
```

```
results = np.argmax(y_pred,axis = 1)
results
```

```
array([77, 50, 80, ..., 63,  0, 22])
```

Trying modifying LeNet architecture

Increase batch size

```
model = Sequential([
# Layer 1
Conv2D(filters = 32, kernel_size = 5, strides = 1, activation =
        'relu', input_shape = (32,32,1),
        kernel_regularizer=regularizers.l1_l2(l1=0, l2=0.0005)),
# Layer 2
Conv2D(filters = 32, kernel_size = 5, strides = 1, use_bias=False),
# Layer 3
BatchNormalization(),
# — — — — — — — — — — — — — — — — #
Activation('relu'),
MaxPooling2D(pool_size = 2, strides = 2),
```

```
Dropout(0.25),
# ------------------- #
# Layer 3
Conv2D(filters = 64, kernel_size = 3, strides = 1, activation =
        'relu', kernel_regularizer=regularizers.l1_l2(l1=0,
        l2=0.0005)),
# Layer 4
Conv2D(filters = 64, kernel_size = 3, strides = 1, use_bias=False),
# Layer 5
BatchNormalization(),
# ------------------- #
Activation('relu'),
MaxPooling2D(pool_size = 2, strides = 2),
Dropout(0.25),
Flatten(),
# ------------------- #
# Layer 6
Dense(units = 320, use_bias=False),
# Layer 7
BatchNormalization(),
# ------------------- #
Activation('relu'),
# ------------------- #
# Layer 8
Dense(units = 160, use_bias=False),
# Layer 9
BatchNormalization(),
# ------------------- #
Activation('relu'),
# ------------------- #
# Layer 10
Dense(units = 128, use_bias=False),
# Layer 11
BatchNormalization(),
# ------------------- #
Activation('relu'),
Dropout(0.25),
# ------------------- #
# Output
Dense(units = 100, activation = 'softmax')
])
```

```
early_stopping = EarlyStopping(min_delta = 0.001,patience =
        20,restore_best_weights = True,verbose = 0)
model.compile(optimizer='adam', loss='categorical_crossentropy',
        metrics=['accuracy'])

# model.fit(X_train_re, y_train, batch_size = 400, epochs =
        50,callbacks = [early_stopping], validation_data = (X_test_re,
        y_test))
history = model.fit(X_train_re, y_train, epochs = 50, batch_size =
        150, callbacks = [early_stopping], validation_data =
        (X_test_re, y_test))
```

```
Epoch 1/50
2500/2500 [==============================] - 41s 13ms/step - loss:
2.2116 - accuracy: 0.4587 - val_loss: 1.5957 - val_accuracy: 0.5915
Epoch 2/50
2500/2500 [==============================] - 32s 13ms/step - loss:
1.5810 - accuracy: 0.5981 - val_loss: 1.3743 - val_accuracy: 0.6450
Epoch 3/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.4423 - accuracy: 0.6323 - val_loss: 1.2739 - val_accuracy: 0.6710
Epoch 4/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.3668 - accuracy: 0.6505 - val_loss: 1.1954 - val_accuracy: 0.6905
Epoch 5/50
2500/2500 [==============================] - 31s 12ms/step - loss:
1.3173 - accuracy: 0.6633 - val_loss: 1.2172 - val_accuracy: 0.6856
Epoch 6/50
2500/2500 [==============================] - 29s 12ms/step - loss:
1.2796 - accuracy: 0.6715 - val_loss: 1.1713 - val_accuracy: 0.6970
Epoch 7/50
2500/2500 [==============================] - 31s 12ms/step - loss:
1.2501 - accuracy: 0.6787 - val_loss: 1.1428 - val_accuracy: 0.7053
Epoch 8/50
2500/2500 [==============================] - 31s 12ms/step - loss:
1.2269 - accuracy: 0.6847 - val_loss: 1.1322 - val_accuracy: 0.7077
Epoch 9/50
2500/2500 [==============================] - 29s 12ms/step - loss:
1.2061 - accuracy: 0.6898 - val_loss: 1.1290 - val_accuracy: 0.7090
Epoch 10/50
2500/2500 [==============================] - 31s 12ms/step - loss:
1.1898 - accuracy: 0.6943 - val_loss: 1.0969 - val_accuracy: 0.7168
Epoch 11/50
```

```
2500/2500 [==============================] - 31s 12ms/step - loss:
1.1772 - accuracy: 0.6967 - val_loss: 1.1215 - val_accuracy: 0.7130
Epoch 12/50
2500/2500 [==============================] - 29s 12ms/step - loss:
1.1615 - accuracy: 0.7003 - val_loss: 1.0929 - val_accuracy: 0.7197
Epoch 13/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.1515 - accuracy: 0.7030 - val_loss: 1.0917 - val_accuracy: 0.7193
Epoch 14/50
2500/2500 [==============================] - 31s 12ms/step - loss:
1.1431 - accuracy: 0.7050 - val_loss: 1.0779 - val_accuracy: 0.7221
Epoch 15/50
2500/2500 [==============================] - 31s 13ms/step - loss:
1.1311 - accuracy: 0.7085 - val_loss: 1.0904 - val_accuracy: 0.7206
Epoch 16/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.1220 - accuracy: 0.7108 - val_loss: 1.0715 - val_accuracy: 0.7239
Epoch 17/50
2500/2500 [==============================] - 31s 12ms/step - loss:
1.1153 - accuracy: 0.7110 - val_loss: 1.0677 - val_accuracy: 0.7280
Epoch 18/50
2500/2500 [==============================] - 32s 13ms/step - loss:
1.1077 - accuracy: 0.7131 - val_loss: 1.0531 - val_accuracy: 0.7295
Epoch 19/50
2500/2500 [==============================] - 31s 12ms/step - loss:
1.1004 - accuracy: 0.7154 - val_loss: 1.0938 - val_accuracy: 0.7196
Epoch 20/50
2500/2500 [==============================] - 29s 12ms/step - loss:
1.0953 - accuracy: 0.7169 - val_loss: 1.0504 - val_accuracy: 0.7322
Epoch 21/50
2500/2500 [==============================] - 29s 12ms/step - loss:
1.0891 - accuracy: 0.7175 - val_loss: 1.0610 - val_accuracy: 0.7271
Epoch 22/50
2500/2500 [==============================] - 31s 12ms/step - loss:
1.0817 - accuracy: 0.7195 - val_loss: 1.0567 - val_accuracy: 0.7311
Epoch 23/50
2500/2500 [==============================] - 29s 12ms/step - loss:
1.0790 - accuracy: 0.7194 - val_loss: 1.0614 - val_accuracy: 0.7284
Epoch 24/50
2500/2500 [==============================] - 29s 12ms/step - loss:
```

```
1.0723 - accuracy: 0.7222 - val_loss: 1.0620 - val_accuracy: 0.7291
Epoch 25/50
2500/2500 [==============================] - 31s 12ms/step - loss:
1.0685 - accuracy: 0.7226 - val_loss: 1.0496 - val_accuracy: 0.7308
Epoch 26/50
2500/2500 [==============================] - 29s 12ms/step - loss:
1.0643 - accuracy: 0.7239 - val_loss: 1.0525 - val_accuracy: 0.7322
Epoch 27/50
2500/2500 [==============================] - 29s 12ms/step - loss:
1.0615 - accuracy: 0.7247 - val_loss: 1.0622 - val_accuracy: 0.7284
Epoch 28/50
2500/2500 [==============================] - 32s 13ms/step - loss:
1.0564 - accuracy: 0.7258 - val_loss: 1.0464 - val_accuracy: 0.7344
Epoch 29/50
2500/2500 [==============================] - 31s 13ms/step - loss:
1.0523 - accuracy: 0.7264 - val_loss: 1.0464 - val_accuracy: 0.7329
Epoch 30/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.0504 - accuracy: 0.7274 - val_loss: 1.0430 - val_accuracy: 0.7353
Epoch 31/50
2500/2500 [==============================] - 32s 13ms/step - loss:
1.0470 - accuracy: 0.7276 - val_loss: 1.0809 - val_accuracy: 0.7252
Epoch 32/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.0410 - accuracy: 0.7291 - val_loss: 1.0540 - val_accuracy: 0.7315
Epoch 33/50
2500/2500 [==============================] - 31s 12ms/step - loss:
1.0401 - accuracy: 0.7293 - val_loss: 1.0729 - val_accuracy: 0.7264
Epoch 34/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.0361 - accuracy: 0.7307 - val_loss: 1.0624 - val_accuracy: 0.7309
Epoch 35/50
2500/2500 [==============================] - 32s 13ms/step - loss:
1.0335 - accuracy: 0.7311 - val_loss: 1.0749 - val_accuracy: 0.7274
Epoch 36/50
2500/2500 [==============================] - 31s 12ms/step - loss:
1.0308 - accuracy: 0.7319 - val_loss: 1.0504 - val_accuracy: 0.7331
Epoch 37/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.0275 - accuracy: 0.7329 - val_loss: 1.0430 - val_accuracy: 0.7358
```

```
Epoch 38/50
2500/2500 [==============================] - 31s 12ms/step - loss:
1.0257 - accuracy: 0.7324 - val_loss: 1.0396 - val_accuracy: 0.7358
Epoch 39/50
2500/2500 [==============================] - 31s 12ms/step - loss:
1.0230 - accuracy: 0.7340 - val_loss: 1.0398 - val_accuracy: 0.7369
Epoch 40/50
2500/2500 [==============================] - 31s 12ms/step - loss:
1.0236 - accuracy: 0.7333 - val_loss: 1.0351 - val_accuracy: 0.7371
Epoch 41/50
2500/2500 [==============================] - 31s 12ms/step - loss:
1.0172 - accuracy: 0.7352 - val_loss: 1.0561 - val_accuracy: 0.7349
Epoch 42/50
2500/2500 [==============================] - 31s 13ms/step - loss:
1.0153 - accuracy: 0.7355 - val_loss: 1.0379 - val_accuracy: 0.7378
Epoch 43/50
2500/2500 [==============================] - 29s 12ms/step - loss:
1.0141 - accuracy: 0.7355 - val_loss: 1.0442 - val_accuracy: 0.7354
Epoch 44/50
2500/2500 [==============================] - 31s 12ms/step - loss:
1.0114 - accuracy: 0.7354 - val_loss: 1.0383 - val_accuracy: 0.7360
Epoch 45/50
2500/2500 [==============================] - 29s 12ms/step - loss:
1.0093 - accuracy: 0.7362 - val_loss: 1.0398 - val_accuracy: 0.7371
Epoch 46/50
2500/2500 [==============================] - 31s 13ms/step - loss:
1.0104 - accuracy: 0.7360 - val_loss: 1.0411 - val_accuracy: 0.7377
Epoch 47/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.0043 - accuracy: 0.7384 - val_loss: 1.0775 - val_accuracy: 0.7279
Epoch 48/50
2500/2500 [==============================] - 31s 13ms/step - loss:
1.0028 - accuracy: 0.7380 - val_loss: 1.0375 - val_accuracy: 0.7366
Epoch 49/50
2500/2500 [==============================] - 32s 13ms/step - loss:
1.0010 - accuracy: 0.7383 - val_loss: 1.0387 - val_accuracy: 0.7370
Epoch 50/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.0015 - accuracy: 0.7386 - val_loss: 1.0338 - val_accuracy: 0.7378
```

Trying to change neurons in hidden layers

```
model = Sequential([
# Layer 1
Conv2D(filters = 32, kernel_size = 5, strides = 1, activation =
        'relu', input_shape = (32,32,1),
        kernel_regularizer=regularizers.l1_l2(l1=0, l2=0.0005)),
# Layer 2
Conv2D(filters = 32, kernel_size = 5, strides = 1, use_bias=False),
# Layer 3
BatchNormalization(),
# — — — — — — — — — — — — — — — — #
Activation('relu'),
MaxPooling2D(pool_size = 2, strides = 2),
Dropout(0.25),
# — — — — — — — — — — — — — — — — #
# Layer 3
Conv2D(filters = 64, kernel_size = 3, strides = 1, activation =
        'relu', kernel_regularizer=regularizers.l1_l2(l1=0,
        l2=0.0005)),
# Layer 4
Conv2D(filters = 64, kernel_size = 3, strides = 1, use_bias=False),
# Layer 5
BatchNormalization(),
# — — — — — — — — — — — — — — — — #
Activation('relu'),
MaxPooling2D(pool_size = 2, strides = 2),
Dropout(0.25),
Flatten(),
# — — — — — — — — — — — — — — — — #
# Layer 6
Dense(units = 256, use_bias=False),
# Layer 7
BatchNormalization(),
# — — — — — — — — — — — — — — — — #
Activation('relu'),
# — — — — — — — — — — — — — — — — #
# Layer 10
Dense(units = 128, use_bias=False),
# Layer 11
BatchNormalization(),
# — — — — — — — — — — — — — — — — #
```

```
Activation('relu'),
Dropout(0.25),
# - - - - - - - - - - - - - - - - - #
# Output
Dense(units = 100, activation = 'softmax')
])

early_stopping = EarlyStopping(min_delta = 0.001,patience =
        20,restore_best_weights = True,verbose = 0)
model.compile(optimizer='adam', loss='categorical_crossentropy',
        metrics=['accuracy'])


# model.fit(X_train_re, y_train, batch_size = 400, epochs =
        50,callbacks = [early_stopping], validation_data = (X_test_re,
        y_test))
history = model.fit(X_train_re, y_train, epochs = 50, batch_size =
        150, callbacks = [early_stopping], validation_data =
        (X_test_re, y_test))
```

```
Epoch 1/50
2500/2500 [==============================] - 31s 12ms/step - loss:
2.2103 - accuracy: 0.4587 - val_loss: 1.6014 - val_accuracy: 0.5920
Epoch 2/50
2500/2500 [==============================] - 29s 12ms/step - loss:
1.5995 - accuracy: 0.5932 - val_loss: 1.3521 - val_accuracy: 0.6519
Epoch 3/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.4686 - accuracy: 0.6246 - val_loss: 1.3037 - val_accuracy: 0.6650
Epoch 4/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.3992 - accuracy: 0.6420 - val_loss: 1.3229 - val_accuracy: 0.6599
Epoch 5/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.3533 - accuracy: 0.6536 - val_loss: 1.2024 - val_accuracy: 0.6896
Epoch 6/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.3196 - accuracy: 0.6615 - val_loss: 1.1897 - val_accuracy: 0.6926
Epoch 7/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.2899 - accuracy: 0.6685 - val_loss: 1.1597 - val_accuracy: 0.7022
Epoch 8/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.2709 - accuracy: 0.6730 - val_loss: 1.1717 - val_accuracy: 0.6966
```

```
Epoch 9/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.2518 - accuracy: 0.6780 - val_loss: 1.1402 - val_accuracy: 0.7075
Epoch 10/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.2366 - accuracy: 0.6815 - val_loss: 1.1636 - val_accuracy: 0.7006
Epoch 11/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.2245 - accuracy: 0.6843 - val_loss: 1.1111 - val_accuracy: 0.7146
Epoch 12/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.2137 - accuracy: 0.6867 - val_loss: 1.1004 - val_accuracy: 0.7170
Epoch 13/50
2500/2500 [==============================] - 29s 11ms/step - loss:
1.2040 - accuracy: 0.6897 - val_loss: 1.1071 - val_accuracy: 0.7157
Epoch 14/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.1941 - accuracy: 0.6921 - val_loss: 1.1185 - val_accuracy: 0.7122
Epoch 15/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.1865 - accuracy: 0.6937 - val_loss: 1.1260 - val_accuracy: 0.7110
Epoch 16/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.1797 - accuracy: 0.6949 - val_loss: 1.1019 - val_accuracy: 0.7187
Epoch 17/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.1733 - accuracy: 0.6971 - val_loss: 1.0888 - val_accuracy: 0.7215
Epoch 18/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.1659 - accuracy: 0.6986 - val_loss: 1.1507 - val_accuracy: 0.7043
Epoch 19/50
2500/2500 [==============================] - 29s 11ms/step - loss:
1.1612 - accuracy: 0.7001 - val_loss: 1.0709 - val_accuracy: 0.7264
Epoch 20/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.1545 - accuracy: 0.7015 - val_loss: 1.0786 - val_accuracy: 0.7232
Epoch 21/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.1490 - accuracy: 0.7021 - val_loss: 1.0697 - val_accuracy: 0.7243
Epoch 22/50
```

```
2500/2500 [==============================] - 28s 11ms/step - loss:
1.1475 - accuracy: 0.7032 - val_loss: 1.1085 - val_accuracy: 0.7174
Epoch 23/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.1388 - accuracy: 0.7055 - val_loss: 1.0930 - val_accuracy: 0.7198
Epoch 24/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.1383 - accuracy: 0.7050 - val_loss: 1.0703 - val_accuracy: 0.7261
Epoch 25/50
2500/2500 [==============================] - 29s 11ms/step - loss:
1.1337 - accuracy: 0.7065 - val_loss: 1.0836 - val_accuracy: 0.7227
Epoch 26/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.1295 - accuracy: 0.7073 - val_loss: 1.0606 - val_accuracy: 0.7294
Epoch 27/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.1251 - accuracy: 0.7086 - val_loss: 1.0879 - val_accuracy: 0.7215
Epoch 28/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.1241 - accuracy: 0.7083 - val_loss: 1.0568 - val_accuracy: 0.7302
Epoch 29/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.1203 - accuracy: 0.7097 - val_loss: 1.0652 - val_accuracy: 0.7285
Epoch 30/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.1182 - accuracy: 0.7096 - val_loss: 1.1191 - val_accuracy: 0.7119
Epoch 31/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.1151 - accuracy: 0.7108 - val_loss: 1.0513 - val_accuracy: 0.7318
Epoch 32/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.1112 - accuracy: 0.7114 - val_loss: 1.0834 - val_accuracy: 0.7229
Epoch 33/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.1097 - accuracy: 0.7122 - val_loss: 1.0634 - val_accuracy: 0.7276
Epoch 34/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.1073 - accuracy: 0.7128 - val_loss: 1.0734 - val_accuracy: 0.7251
Epoch 35/50
2500/2500 [==============================] - 30s 12ms/step - loss:
```

```
1.1040 - accuracy: 0.7125 - val_loss: 1.0718 - val_accuracy: 0.7261
Epoch 36/50
2500/2500 [==============================] - 29s 11ms/step - loss:
1.1036 - accuracy: 0.7127 - val_loss: 1.0558 - val_accuracy: 0.7292
Epoch 37/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.1017 - accuracy: 0.7144 - val_loss: 1.1079 - val_accuracy: 0.7157
Epoch 38/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.0992 - accuracy: 0.7150 - val_loss: 1.0574 - val_accuracy: 0.7302
Epoch 39/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.0958 - accuracy: 0.7158 - val_loss: 1.0509 - val_accuracy: 0.7309
Epoch 40/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.0946 - accuracy: 0.7155 - val_loss: 1.1161 - val_accuracy: 0.7151
Epoch 41/50
2500/2500 [==============================] - 29s 12ms/step - loss:
1.0942 - accuracy: 0.7159 - val_loss: 1.0506 - val_accuracy: 0.7329
Epoch 42/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.0900 - accuracy: 0.7163 - val_loss: 1.0706 - val_accuracy: 0.7265
Epoch 43/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.0888 - accuracy: 0.7164 - val_loss: 1.0460 - val_accuracy: 0.7339
Epoch 44/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.0866 - accuracy: 0.7170 - val_loss: 1.0615 - val_accuracy: 0.7298
Epoch 45/50
2500/2500 [==============================] - 29s 11ms/step - loss:
1.0852 - accuracy: 0.7171 - val_loss: 1.0690 - val_accuracy: 0.7258
Epoch 46/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.0855 - accuracy: 0.7179 - val_loss: 1.0500 - val_accuracy: 0.7329
Epoch 47/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.0812 - accuracy: 0.7188 - val_loss: 1.0631 - val_accuracy: 0.7292
Epoch 48/50
2500/2500 [==============================] - 30s 12ms/step - loss:
1.0832 - accuracy: 0.7182 - val_loss: 1.0621 - val_accuracy: 0.7305
```

```
Epoch 49/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.0792 - accuracy: 0.7190 - val_loss: 1.0559 - val_accuracy: 0.7306
Epoch 50/50
2500/2500 [==============================] - 28s 11ms/step - loss:
1.0749 - accuracy: 0.7199 - val_loss: 1.0655 - val_accuracy: 0.7294
```

```
save_path = '/content/drive/MyDrive/AML mini
        project/lenet_50e_150b_2.h5'
model.save(save_path,save_format='tf')
```

Reduce batch size back to 100, with new architecture(modified hidden layers)

```
model = Sequential([
# Layer 1
Conv2D(filters = 32, kernel_size = 5, strides = 1, activation =
        'relu', input_shape = (32,32,1),
        kernel_regularizer=regularizers.l1_l2(l1=0, l2=0.0005)),
# Layer 2
Conv2D(filters = 32, kernel_size = 5, strides = 1, use_bias=False),
# Layer 3
BatchNormalization(),
# — — — — — — — — — — — — — — — — — #
Activation('relu'),
MaxPooling2D(pool_size = 2, strides = 2),
Dropout(0.25),
# — — — — — — — — — — — — — — — — — #
# Layer 3
Conv2D(filters = 64, kernel_size = 3, strides = 1, activation =
        'relu', kernel_regularizer=regularizers.l1_l2(l1=0,
        l2=0.0005)),
# Layer 4
Conv2D(filters = 64, kernel_size = 3, strides = 1, use_bias=False),
# Layer 5
BatchNormalization(),
# — — — — — — — — — — — — — — — — — #
Activation('relu'),
MaxPooling2D(pool_size = 2, strides = 2),
Dropout(0.25),
Flatten(),
# — — — — — — — — — — — — — — — — — #
# Layer 6
```

```
Dense(units = 256, use_bias=False),
# Layer 7
BatchNormalization(),
# - - - - - - - - - - - - - - - - #
Activation('relu'),
# - - - - - - - - - - - - - - - - #
# Layer 10
Dense(units = 128, use_bias=False),
# Layer 11
BatchNormalization(),
# - - - - - - - - - - - - - - - - #
Activation('relu'),
Dropout(0.25),
# - - - - - - - - - - - - - - - - #
# Output
Dense(units = 100, activation = 'softmax')
])

early_stopping = EarlyStopping(min_delta = 0.001,patience =
        20,restore_best_weights = True,verbose = 0)
model.compile(optimizer='adam', loss='categorical_crossentropy',
        metrics=['accuracy'])


# model.fit(X_train_re, y_train, batch_size = 400, epochs =
        50,callbacks = [early_stopping], validation_data = (X_test_re,
        y_test))
history = model.fit(X_train_re, y_train, epochs = 50, batch_size =
        100, callbacks = [early_stopping], validation_data =
        (X_test_re, y_test))
```

```
Epoch 1/50
3750/3750 [==============================] - 43s 9ms/step - loss:
2.1532 - accuracy: 0.4711 - val_loss: 1.5326 - val_accuracy: 0.6105
Epoch 2/50
3750/3750 [==============================] - 32s 8ms/step - loss:
1.6008 - accuracy: 0.5934 - val_loss: 1.3645 - val_accuracy: 0.6505
Epoch 3/50
3750/3750 [==============================] - 31s 8ms/step - loss:
1.4787 - accuracy: 0.6228 - val_loss: 1.2879 - val_accuracy: 0.6712
Epoch 4/50
3750/3750 [==============================] - 31s 8ms/step - loss:
1.4118 - accuracy: 0.6392 - val_loss: 1.2095 - val_accuracy: 0.6888
Epoch 5/50
```

```
3750/3750 [==============================] - 33s 9ms/step - loss:
1.3690 - accuracy: 0.6508 - val_loss: 1.1964 - val_accuracy: 0.6921
Epoch 6/50
3750/3750 [==============================] - 31s 8ms/step - loss:
1.3377 - accuracy: 0.6578 - val_loss: 1.1704 - val_accuracy: 0.6999
Epoch 7/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.3142 - accuracy: 0.6641 - val_loss: 1.1578 - val_accuracy: 0.7024
Epoch 8/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.2944 - accuracy: 0.6688 - val_loss: 1.1562 - val_accuracy: 0.7045
Epoch 9/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.2781 - accuracy: 0.6729 - val_loss: 1.1708 - val_accuracy: 0.6998
Epoch 10/50
3750/3750 [==============================] - 31s 8ms/step - loss:
1.2610 - accuracy: 0.6775 - val_loss: 1.1177 - val_accuracy: 0.7135
Epoch 11/50
3750/3750 [==============================] - 31s 8ms/step - loss:
1.2495 - accuracy: 0.6789 - val_loss: 1.1286 - val_accuracy: 0.7114
Epoch 12/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.2407 - accuracy: 0.6816 - val_loss: 1.1106 - val_accuracy: 0.7158
Epoch 13/50
3750/3750 [==============================] - 31s 8ms/step - loss:
1.2291 - accuracy: 0.6844 - val_loss: 1.1114 - val_accuracy: 0.7157
Epoch 14/50
3750/3750 [==============================] - 31s 8ms/step - loss:
1.2231 - accuracy: 0.6861 - val_loss: 1.1042 - val_accuracy: 0.7178
Epoch 15/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.2136 - accuracy: 0.6883 - val_loss: 1.0939 - val_accuracy: 0.7197
Epoch 16/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.2075 - accuracy: 0.6899 - val_loss: 1.1110 - val_accuracy: 0.7162
Epoch 17/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1975 - accuracy: 0.6922 - val_loss: 1.1220 - val_accuracy: 0.7115
Epoch 18/50
3750/3750 [==============================] - 32s 8ms/step - loss:
```

```
1.1949 - accuracy: 0.6927 - val_loss: 1.0900 - val_accuracy: 0.7219
Epoch 19/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1884 - accuracy: 0.6938 - val_loss: 1.0867 - val_accuracy: 0.7224
Epoch 20/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1837 - accuracy: 0.6952 - val_loss: 1.0910 - val_accuracy: 0.7215
Epoch 21/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1786 - accuracy: 0.6965 - val_loss: 1.0993 - val_accuracy: 0.7205
Epoch 22/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1745 - accuracy: 0.6973 - val_loss: 1.0767 - val_accuracy: 0.7240
Epoch 23/50
3750/3750 [==============================] - 32s 8ms/step - loss:
1.1700 - accuracy: 0.6984 - val_loss: 1.0681 - val_accuracy: 0.7272
Epoch 24/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1681 - accuracy: 0.6988 - val_loss: 1.1138 - val_accuracy: 0.7176
Epoch 25/50
3750/3750 [==============================] - 32s 8ms/step - loss:
1.1628 - accuracy: 0.7006 - val_loss: 1.0797 - val_accuracy: 0.7258
Epoch 26/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1576 - accuracy: 0.7022 - val_loss: 1.0691 - val_accuracy: 0.7264
Epoch 27/50
3750/3750 [==============================] - 32s 8ms/step - loss:
1.1582 - accuracy: 0.7014 - val_loss: 1.0788 - val_accuracy: 0.7257
Epoch 28/50
3750/3750 [==============================] - 31s 8ms/step - loss:
1.1535 - accuracy: 0.7026 - val_loss: 1.0734 - val_accuracy: 0.7254
Epoch 29/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1483 - accuracy: 0.7038 - val_loss: 1.0970 - val_accuracy: 0.7198
Epoch 30/50
3750/3750 [==============================] - 31s 8ms/step - loss:
1.1466 - accuracy: 0.7040 - val_loss: 1.0746 - val_accuracy: 0.7260
Epoch 31/50
3750/3750 [==============================] - 31s 8ms/step - loss:
1.1456 - accuracy: 0.7052 - val_loss: 1.0736 - val_accuracy: 0.7265
```

```
Epoch 32/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1435 - accuracy: 0.7048 - val_loss: 1.0736 - val_accuracy: 0.7256
Epoch 33/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1372 - accuracy: 0.7068 - val_loss: 1.0707 - val_accuracy: 0.7278
Epoch 34/50
3750/3750 [==============================] - 31s 8ms/step - loss:
1.1367 - accuracy: 0.7071 - val_loss: 1.0758 - val_accuracy: 0.7268
Epoch 35/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1344 - accuracy: 0.7072 - val_loss: 1.0673 - val_accuracy: 0.7304
Epoch 36/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1315 - accuracy: 0.7078 - val_loss: 1.0574 - val_accuracy: 0.7305
Epoch 37/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1320 - accuracy: 0.7083 - val_loss: 1.0718 - val_accuracy: 0.7268
Epoch 38/50
3750/3750 [==============================] - 32s 8ms/step - loss:
1.1276 - accuracy: 0.7085 - val_loss: 1.0619 - val_accuracy: 0.7291
Epoch 39/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1264 - accuracy: 0.7093 - val_loss: 1.0466 - val_accuracy: 0.7337
Epoch 40/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1259 - accuracy: 0.7098 - val_loss: 1.0781 - val_accuracy: 0.7258
Epoch 41/50
3750/3750 [==============================] - 31s 8ms/step - loss:
1.1213 - accuracy: 0.7106 - val_loss: 1.0522 - val_accuracy: 0.7309
Epoch 42/50
3750/3750 [==============================] - 31s 8ms/step - loss:
1.1213 - accuracy: 0.7106 - val_loss: 1.0706 - val_accuracy: 0.7276
Epoch 43/50
3750/3750 [==============================] - 32s 8ms/step - loss:
1.1195 - accuracy: 0.7111 - val_loss: 1.0562 - val_accuracy: 0.7310
Epoch 44/50
3750/3750 [==============================] - 31s 8ms/step - loss:
1.1164 - accuracy: 0.7110 - val_loss: 1.0514 - val_accuracy: 0.7319
Epoch 45/50
```

```
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1146 - accuracy: 0.7119 - val_loss: 1.0632 - val_accuracy: 0.7295
Epoch 46/50
3750/3750 [==============================] - 31s 8ms/step - loss:
1.1143 - accuracy: 0.7117 - val_loss: 1.0546 - val_accuracy: 0.7326
Epoch 47/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1110 - accuracy: 0.7132 - val_loss: 1.0881 - val_accuracy: 0.7243
Epoch 48/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1132 - accuracy: 0.7127 - val_loss: 1.0584 - val_accuracy: 0.7306
Epoch 49/50
3750/3750 [==============================] - 32s 8ms/step - loss:
1.1096 - accuracy: 0.7133 - val_loss: 1.0570 - val_accuracy: 0.7328
Epoch 50/50
3750/3750 [==============================] - 31s 8ms/step - loss:
1.1082 - accuracy: 0.7135 - val_loss: 1.0591 - val_accuracy: 0.7299
```

```
save_path = '/content/drive/MyDrive/AML mini
        project/lenet_50e_100b_1.h5'
model.save(save_path,save_format='tf')
```

Adding one more hidden layer

```
model = Sequential([
# Layer 1
Conv2D(filters = 32, kernel_size = 5, strides = 1, activation =
        'relu', input_shape = (32,32,1),
        kernel_regularizer=regularizers.l1_l2(l1=0, l2=0.0005)),
# Layer 2
Conv2D(filters = 32, kernel_size = 5, strides = 1, use_bias=False),
# Layer 3
BatchNormalization(),
# — — — — — — — — — — — — — — — — — #
Activation('relu'),
MaxPooling2D(pool_size = 2, strides = 2),
Dropout(0.25),
# — — — — — — — — — — — — — — — — — #
# Layer 3
Conv2D(filters = 64, kernel_size = 3, strides = 1, activation =
        'relu', kernel_regularizer=regularizers.l1_l2(l1=0,
        l2=0.0005)),
# Layer 4
```

```
    Conv2D(filters = 64, kernel_size = 3, strides = 1, use_bias=False),
    # Layer 5
    BatchNormalization(),
    # — — — — — — — — — — — — — — — — — #
    Activation('relu'),
    MaxPooling2D(pool_size = 2, strides = 2),
    Dropout(0.25),
    Flatten(),
    # — — — — — — — — — — — — — — — — — #
    # Layer 6
    Dense(units = 256, use_bias=False),
    # Layer 7
    BatchNormalization(),
    # — — — — — — — — — — — — — — — — — #
    Activation('relu'),
    # — — — — — — — — — — — — — — — — — #
    # Layer 8
    Dense(units = 186, use_bias=False),
    # Layer 9
    BatchNormalization(),
    # — — — — — — — — — — — — — — — — — #
    Activation('relu'),
    # — — — — — — — — — — — — — — — — — #
    # Layer 10
    Dense(units = 128, use_bias=False),
    # Layer 11
    BatchNormalization(),
    # — — — — — — — — — — — — — — — — — #
    Activation('relu'),
    # — — — — — — — — — — — — — — — — — #
    # Layer 12
    Dense(units = 84, use_bias=False),
    # Layer 13
    BatchNormalization(),
    # — — — — — — — — — — — — — — — — — #
    Activation('relu'),
    Dropout(0.25),
    # — — — — — — — — — — — — — — — — — #
    # Output
```

```
Dense(units = 100, activation = 'softmax')
])

early_stopping = EarlyStopping(min_delta = 0.001,patience =
        20,restore_best_weights = True,verbose = 0)
model.compile(optimizer='adam', loss='categorical_crossentropy',
        metrics=['accuracy'])

# model.fit(X_train_re, y_train, batch_size = 400, epochs =
        50,callbacks = [early_stopping], validation_data = (X_test_re,
        y_test))
history = model.fit(X_train_re, y_train, epochs = 50, batch_size =
        100, callbacks = [early_stopping], validation_data =
        (X_test_re, y_test))
```

```
Epoch 1/50
3750/3750 [==============================] - 38s 10ms/step - loss:
2.3009 - accuracy: 0.4401 - val_loss: 1.5902 - val_accuracy: 0.5925
Epoch 2/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.6725 - accuracy: 0.5796 - val_loss: 1.4001 - val_accuracy: 0.6415
Epoch 3/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.5302 - accuracy: 0.6150 - val_loss: 1.3544 - val_accuracy: 0.6550
Epoch 4/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.4518 - accuracy: 0.6347 - val_loss: 1.3184 - val_accuracy: 0.6616
Epoch 5/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.3979 - accuracy: 0.6478 - val_loss: 1.2264 - val_accuracy: 0.6860
Epoch 6/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.3604 - accuracy: 0.6581 - val_loss: 1.1936 - val_accuracy: 0.6949
Epoch 7/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.3326 - accuracy: 0.6646 - val_loss: 1.1751 - val_accuracy: 0.6995
Epoch 8/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.3076 - accuracy: 0.6700 - val_loss: 1.1685 - val_accuracy: 0.7021
Epoch 9/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.2868 - accuracy: 0.6753 - val_loss: 1.1796 - val_accuracy: 0.6988
Epoch 10/50
3750/3750 [==============================] - 34s 9ms/step - loss:
```

```
1.2699 - accuracy: 0.6797 - val_loss: 1.2019 - val_accuracy: 0.6948
Epoch 11/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.2514 - accuracy: 0.6844 - val_loss: 1.1396 - val_accuracy: 0.7086
Epoch 12/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.2419 - accuracy: 0.6866 - val_loss: 1.1343 - val_accuracy: 0.7117
Epoch 13/50
3750/3750 [==============================] - 35s 9ms/step - loss:
1.2315 - accuracy: 0.6892 - val_loss: 1.1484 - val_accuracy: 0.7083
Epoch 14/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.2181 - accuracy: 0.6925 - val_loss: 1.1194 - val_accuracy: 0.7162
Epoch 15/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.2098 - accuracy: 0.6940 - val_loss: 1.1274 - val_accuracy: 0.7134
Epoch 16/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.2026 - accuracy: 0.6949 - val_loss: 1.1141 - val_accuracy: 0.7166
Epoch 17/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1927 - accuracy: 0.6979 - val_loss: 1.1511 - val_accuracy: 0.7071
Epoch 18/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1854 - accuracy: 0.6999 - val_loss: 1.1378 - val_accuracy: 0.7115
Epoch 19/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1802 - accuracy: 0.7009 - val_loss: 1.1031 - val_accuracy: 0.7220
Epoch 20/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1708 - accuracy: 0.7042 - val_loss: 1.1022 - val_accuracy: 0.7199
Epoch 21/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1653 - accuracy: 0.7049 - val_loss: 1.0740 - val_accuracy: 0.7275
Epoch 22/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1634 - accuracy: 0.7052 - val_loss: 1.0928 - val_accuracy: 0.7229
Epoch 23/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1569 - accuracy: 0.7069 - val_loss: 1.0736 - val_accuracy: 0.7268
```

```
Epoch 24/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1520 - accuracy: 0.7077 - val_loss: 1.1143 - val_accuracy: 0.7171
Epoch 25/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.1458 - accuracy: 0.7092 - val_loss: 1.0754 - val_accuracy: 0.7287
Epoch 26/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1428 - accuracy: 0.7105 - val_loss: 1.0879 - val_accuracy: 0.7255
Epoch 27/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.1373 - accuracy: 0.7119 - val_loss: 1.0776 - val_accuracy: 0.7280
Epoch 28/50
3750/3750 [==============================] - 32s 8ms/step - loss:
1.1334 - accuracy: 0.7128 - val_loss: 1.1104 - val_accuracy: 0.7176
Epoch 29/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1299 - accuracy: 0.7135 - val_loss: 1.0899 - val_accuracy: 0.7256
Epoch 30/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.1269 - accuracy: 0.7135 - val_loss: 1.0791 - val_accuracy: 0.7266
Epoch 31/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.1211 - accuracy: 0.7158 - val_loss: 1.0641 - val_accuracy: 0.7316
Epoch 32/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1197 - accuracy: 0.7160 - val_loss: 1.0664 - val_accuracy: 0.7300
Epoch 33/50
3750/3750 [==============================] - 32s 8ms/step - loss:
1.1164 - accuracy: 0.7164 - val_loss: 1.0599 - val_accuracy: 0.7322
Epoch 34/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.1124 - accuracy: 0.7172 - val_loss: 1.0841 - val_accuracy: 0.7273
Epoch 35/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.1100 - accuracy: 0.7181 - val_loss: 1.0623 - val_accuracy: 0.7311
Epoch 36/50
3750/3750 [==============================] - 32s 9ms/step - loss:
1.1069 - accuracy: 0.7182 - val_loss: 1.0741 - val_accuracy: 0.7282
Epoch 37/50
```

```
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1046 - accuracy: 0.7196 - val_loss: 1.0906 - val_accuracy: 0.7242
Epoch 38/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.1035 - accuracy: 0.7191 - val_loss: 1.0509 - val_accuracy: 0.7352
Epoch 39/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.0988 - accuracy: 0.7205 - val_loss: 1.0595 - val_accuracy: 0.7326
Epoch 40/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.0981 - accuracy: 0.7206 - val_loss: 1.0623 - val_accuracy: 0.7338
Epoch 41/50
3750/3750 [==============================] - 35s 9ms/step - loss:
1.0954 - accuracy: 0.7216 - val_loss: 1.1022 - val_accuracy: 0.7211
Epoch 42/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.0918 - accuracy: 0.7229 - val_loss: 1.0681 - val_accuracy: 0.7300
Epoch 43/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.0909 - accuracy: 0.7227 - val_loss: 1.0713 - val_accuracy: 0.7307
Epoch 44/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.0886 - accuracy: 0.7242 - val_loss: 1.0763 - val_accuracy: 0.7291
Epoch 45/50
3750/3750 [==============================] - 35s 9ms/step - loss:
1.0854 - accuracy: 0.7228 - val_loss: 1.0581 - val_accuracy: 0.7340
Epoch 46/50
3750/3750 [==============================] - 34s 9ms/step - loss:
1.0844 - accuracy: 0.7244 - val_loss: 1.0536 - val_accuracy: 0.7346
Epoch 47/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.0824 - accuracy: 0.7244 - val_loss: 1.0541 - val_accuracy: 0.7357
Epoch 48/50
3750/3750 [==============================] - 33s 9ms/step - loss:
1.0825 - accuracy: 0.7242 - val_loss: 1.0521 - val_accuracy: 0.7361
Epoch 49/50
3750/3750 [==============================] - 35s 9ms/step - loss:
1.0792 - accuracy: 0.7255 - val_loss: 1.0494 - val_accuracy: 0.7356
Epoch 50/50
```

```
3750/3750 [==============================] - 35s 9ms/step - loss: 1.0761 - accuracy: 0.7261 - val_loss: 1.0434 - val_accuracy: 0.7368
```

```
save_path = '/content/drive/MyDrive/AML mini
        project/lenet_50e_dense.h5'
model.save(save_path,save_format='tf')
```

```
X_train_re = None
```

Predictions

```
df_test = pickle.load(open(test_path,'rb'))
df_test['data'] = df_test['data']/255
df_test.head()
```

	data
0	[[0.0, 0.0, 0.0, 0.01568627450980392, 0.015686...
1	[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...
2	[[0.0, 0.011764705882352941, 0.0, 0.0, 0.39607...
3	[[0.0, 0.047058823529411764, 0.215686274509803...
4	[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,...

```
test_data = np.array([img[:,:,newaxis] for img in
        np.array(df_test['data'].values)])
test_data.shape
```

```
(100000, 28, 28, 1)
```

```
X_test_data = np.pad(test_data, ((0,0),(2,2),(2,2),(0,0)), 'constant')
X_test_data.shape
```

```
(100000, 32, 32, 1)
```

```
save_path = '/content/drive/MyDrive/AMLminiproject/lenet_75e.h5'
model = tf.keras.models.load_model(save_path)
y_pred = model.predict(X_test_data)
```

```
3125/3125 [==============================] - 119s 38ms/step
```

```
results = np.argmax(y_pred,axis = 1)
results
```

```
array([77, 50, 80, ..., 63,  0, 22])
```

```
with open('/content/drive/MyDrive/AMLminiproject/project_rsheta.txt',
        'w') as file:  # edit here as your username
    file.write('\n'.join(map(str, results)))
    file.flush()
```

```
y_pred = model.predict(X_test_re)
```

```
3907/3907 [==============================] - 10s 2ms/step
```

```
results = np.argmax(y_pred,axis = 1)
results.shape
```

```
(125000,)
```

```
y_test_new = np.argmax(y_test,axis = 1)
y_test_new.shape
```

```
(125000,)
```

```
from sklearn.metrics import accuracy_score


accuracy_score(y_test_new, results)
```

```
0.731992
```

```
y_test_new[:5]
```

```
array([20, 39, 28, 26, 81])
```

```
results[:5]
```

```
array([51, 39, 28, 26, 81])
```