Large scale

I have added epoch results at the end of models, because I have made adjustments on them as I went. I didn’t just test out the five models you see, I probably played around with it hundreds of times and different parameters, and the epoch result might be slightly different if you run them again, but you can see what I have tried playing around with, and some of the stuff that I thought might work didn’t always work.

This was my best result at one point, using model 3 that I built on the code here.

# Epoch 20/20

# 1875/1875 [==============================] - 7s 4ms/step - loss: 0.0999 - accuracy: 0.9656 - val\_loss: 0.2350 - val\_accuracy: 0.9329

I didn’t like this result too much because I am over fitting 96% accuracy on train and only getting 93% on validation data set.

I tried playing around with many other parameters to prevent overfitting, and I was able to achieve that for model4 and model5 as the train and val accuracy was much closer, but I was not able to break 93% accuracy for them. So above is the best result I got so far. The model4 and model 5 results are commented within the code.

I also tried data augmentation, using image generator, you can also see in the code, and I have tried normal data set and augmented data set for every models, but the augmented data set never performed better than the normal data set. I thought they might be able to learn more.

I have a feeling if I keep learning more of the later stable models, like model 4 and model 5, with slower learning rate, batch size, and running them for much longer time and epoch, I feel like it will eventually be higher, but I was not able to get there within the time I had. I like they are not overfitting and train and val accuracies are very very close, which tells me it is actually learning, instead of memorizing.

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, BatchNormalization, Dropout, Flatten, LeakyReLU

from tensorflow.keras.losses import categorical\_crossentropy

from tensorflow.keras.losses import sparse\_categorical\_crossentropy

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.utils import to\_categorical

import matplotlib.pyplot as plt

# Loading in the fashion data set

mnist = tf.keras.datasets.fashion\_mnist

(train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()

train\_images = train\_images.reshape(60000, 28, 28, 1)

test\_images = test\_images.reshape(10000, 28, 28, 1)

train\_images, test\_images = train\_images/255, test\_images/255

# Let's do some DATA AUGMENTATION

# =============================================================================

# # HSV

# import cv2

# # from PIL import Image

# import numpy as np

#

# gfx=[] # to hold the HSV image array

#

# for i in np.arange(0, 100, 1):

# a = cv2.cvtColor(train\_images[i], cv2.COLOR\_BGR2HSV)

# gfx.append(a)

#

# gfx = np.array(gfx)

# train\_images\_HSV = gfx

#

# gfx\_test=[] # to hold the HSV image array

#

# for i in np.arange(0, 100, 1):

#

# b = cv2.cvtColor(test\_images[i], cv2.COLOR\_BGR2HSV)

#

# gfx\_test.append(b)

#

# gfx = np.array(gfx\_test)

# test\_images\_HSV = gfx

# # it becomes tuple, can't work with those later

#

# =============================================================================

# image generator for normal images

X\_train = train\_images

y\_train = train\_labels

X\_val = test\_images

y\_val = test\_labels

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Using ImageDataGenerator to generate images

train\_datagen = ImageDataGenerator(horizontal\_flip = True,

zoom\_range = 0.5, rotation\_range = 30)

val\_datagen = ImageDataGenerator()

# Flowing training images using train\_datagen generator

train\_generator = train\_datagen.flow(x = X\_train, y = y\_train, batch\_size = 64, seed = 42, shuffle = True)

# Flowing validation images using val\_datagen generator

val\_generator = val\_datagen.flow(x = X\_val, y = y\_val, batch\_size = 64, seed = 42, shuffle = True)

# =============================================================================

# running the data augmented images

# history3 = cnn\_model\_class.fit(train\_generator, validation\_data = val\_generator,batch\_size = 32, epochs = 20, verbose = 1)

# =============================================================================

# Baseline model from class

cnn\_model\_class = tf.keras.Sequential([

tf.keras.layers.Conv2D(32, (3,3), activation='relu', input\_shape=(28,28,1)),

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Dropout(0.25),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(128, activation='relu'),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Dense(10, activation='softmax')

])

# To show the layers in Jupyter notebook

# model\_class = cnn\_model\_class()

# model\_class.summary()

# Fitting the model

cnn\_model\_class.compile(loss = 'sparse\_categorical\_crossentropy', optimizer = tf.keras.optimizers.Adam(0.001), metrics = ['accuracy'])

history\_model\_class = cnn\_model\_class.fit(train\_images, train\_labels, batch\_size = 32, verbose = 1, epochs = 20, validation\_data=(test\_images, test\_labels))

# =============================================================================

# # Plotting the accuracies

# dict\_hist = history\_model\_class.history

# list\_ep = [i for i in range(1, 21)]

# plt.figure(figsize = (8, 8))

# plt.plot(list\_ep, dict\_hist['accuracy'], ls = '--', label = 'accuracy')

# plt.plot(list\_ep, dict\_hist['val\_accuracy'], ls = '--', label = 'val\_accuracy')

# plt.ylabel('Accuracy')

# plt.xlabel('Epochs')

# plt.legend()

# plt.show()

#

# =============================================================================

# Using Dropout layers

cnn\_model1 = tf.keras.Sequential([

tf.keras.layers.Conv2D(64, (3,3), activation='relu', input\_shape=(28,28,1)),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Dropout(0.25),

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Dropout(0.25),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(64, activation='relu'),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Dense(64, activation='relu'),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Dense(10, activation='softmax')

])

# Fitting the model

cnn\_model1.compile(loss = 'sparse\_categorical\_crossentropy', optimizer = tf.keras.optimizers.Adam(0.001), metrics = ['accuracy'])

history\_model\_class = cnn\_model1.fit(train\_images, train\_labels, batch\_size = 32, verbose = 1, epochs = 20, validation\_data=(test\_images, test\_labels))

# Using Batchnormalization layers

cnn\_model2 = tf.keras.Sequential([

tf.keras.layers.Conv2D(64, (3,3), activation='relu', input\_shape=(28,28,1)),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(64, activation='relu'),

tf.keras.layers.Dense(64, activation='relu'),

tf.keras.layers.Dense(10, activation='softmax')

])

# Fitting the model

cnn\_model2.compile(loss = 'sparse\_categorical\_crossentropy', optimizer = tf.keras.optimizers.Adam(0.001), metrics = ['accuracy'])

history\_model\_class = cnn\_model2.fit(train\_images, train\_labels, batch\_size = 32, verbose = 1, epochs = 20, validation\_data=(test\_images, test\_labels))

from tensorflow.keras import backend

backend.clear\_session()

# from ddrop.layers import DropConnect, x = DropConnect(Dense(64, activation='relu'), prob=0.5)(x)

# Using Dropout layers, and more filters

cnn\_model3 = tf.keras.Sequential([

tf.keras.layers.Conv2D(128, (3,3), activation='relu', input\_shape=(28,28,1)),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.Conv2D(32, (3,3), activation='relu'),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Dropout(0.25),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(512, activation='relu'),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Dense(32, activation='relu'),

tf.keras.layers.Dropout(0.25),

tf.keras.layers.Dense(10, activation='softmax')

])

# Fitting the model

cnn\_model3.compile(loss = 'sparse\_categorical\_crossentropy', optimizer = tf.keras.optimizers.Adam(0.001), metrics = ['accuracy'])

# on normal image dataset

history\_model\_class = cnn\_model3.fit(train\_images, train\_labels, batch\_size = 32, verbose = 1, epochs = 20, validation\_data=(test\_images, test\_labels))

# Epoch 20/20

# 1875/1875 [==============================] - 7s 4ms/step - loss: 0.0999 - accuracy: 0.9656 - val\_loss: 0.2350 - val\_accuracy: 0.9329

# on augmented data set

history3 = cnn\_model3.fit(train\_generator, validation\_data = val\_generator,batch\_size = 32, epochs = 20, verbose = 1)

# Epoch 20/20

# 750/750 [==============================] - 8s 11ms/step - loss: 0.4361 - accuracy: 0.8450 - val\_loss: 0.3846 - val\_accuracy: 0.8724

# Playing around, drop connection instead of dropout?

# from ddrop.layers import DropConnect

# x = DropConnect(Dense(64, activation='relu'), prob=0.5)(x)

cnn\_model4 = tf.keras.Sequential([

tf.keras.layers.Conv2D(512, (3,3), input\_shape=(28,28,1)),

tf.keras.layers.LeakyReLU(),

tf.keras.layers.Conv2D(512, (3,3), activation='relu'),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Conv2D(512, (3,3), activation='relu'),

tf.keras.layers.Conv2D(512, (3,3), activation='relu'),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Conv2D(256, (3,3), activation='relu'),

tf.keras.layers.Conv2D(256, (3,3), activation='relu'),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Conv2D(128, (3,3), activation='relu'),

tf.keras.layers.Conv2D(128, (3,3), activation='relu'),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.Dropout(0.3),

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Dropout(0.3),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(1048, activation='relu'),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Dense(512, activation='relu'),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Dense(10, activation='softmax')

])

# Fitting the model

adam = tf.keras.optimizers.legacy.Adam(learning\_rate=0.001, decay=1e-6)

cnn\_model4.compile(loss = 'sparse\_categorical\_crossentropy', optimizer = adam, metrics = ['accuracy'])

# on normal image dataset

history\_model\_class = cnn\_model4.fit(train\_images, train\_labels, batch\_size = 32, verbose = 1, epochs = 20, validation\_data=(test\_images, test\_labels))

#Epoch 20/20

#1875/1875 [==============================] - 22s 12ms/step - loss: 0.0284 - accuracy: 0.9898 - val\_loss: 0.3078 - val\_accuracy: 0.9398

# on augmented data set

history4 = cnn\_model4.fit(train\_generator, validation\_data = val\_generator,batch\_size = 32, epochs = 20, verbose = 1)

# for model 5, I was playing around with batch size, and I had to optimize the model and make it not too massive like previous one, because it takes so long

cnn\_model5 = tf.keras.Sequential([

tf.keras.layers.Conv2D(512, (3,3), input\_shape=(28,28,1)),

tf.keras.layers.LeakyReLU(),

tf.keras.layers.Conv2D(512, (3,3), activation='relu'),

tf.keras.layers.BatchNormalization(),

#tf.keras.layers.Dropout(0.4),

tf.keras.layers.Conv2D(128, (3,3), activation='relu'),

tf.keras.layers.Conv2D(128, (3,3), activation='relu'),

tf.keras.layers.BatchNormalization(),

#tf.keras.layers.Dropout(0.3),

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Dropout(0.3),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(1048, activation='relu'),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Dense(512, activation='relu'),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Dense(10, activation='softmax')

])

# Fitting the model

adam = tf.keras.optimizers.legacy.Adam(learning\_rate=0.0001, decay=1e-6)

cnn\_model5.compile(loss = 'sparse\_categorical\_crossentropy', optimizer = adam, metrics = ['accuracy'])

# on normal image dataset

history\_model\_class = cnn\_model5.fit(train\_images, train\_labels, batch\_size = 1024, verbose = 1, epochs = 20, validation\_data=(test\_images, test\_labels))

#Epoch 20/20

#1875/1875 [==============================] - 65s 34ms/step - loss: 0.2069 - accuracy: 0.9247 - val\_loss: 0.2320 - val\_accuracy: 0.9181

history5 = cnn\_model5.fit(train\_generator, validation\_data = val\_generator,batch\_size = 1024, epochs = 30, verbose = 1) #image generator, data augmented