Deep Learning Report - CNN

# Transfer Learning

## Introduction

This project will evaluate a MLP and a series of CNN networks that were build using keras with tensorflow background in python. The models are built using a random seed number generator. This seed is support to be able to be manipulated to have consistent results by using numpy.random.seed(1234), but there is a known issue with using tensorflow backend that does not set this seed preventing us from obtaining consistent results. In order to attempt to bypass this issue, each model was built five times to obtain the average results. All result in this project are averages of five models built.

The MLP model had two input layers using ReLu activation and a classification output layer using softmax activation. The first CNN model was built with a single convolutional layer with 16, 3x3 filters with ReLu activation, a 2x2 max pooling layer, a hidden layer of 128 neurons using ReLu activation, a dropout layer, and a output layer using softmax activation. The other CNN models were build off the first CNN model with the following changes: 2 convolutional layers, 2 convolutional layers and 2 max pooling layers, 32 filters, 5x5 filters, 7x7 filters, no max pooling layer, no dropout layer.

## Results



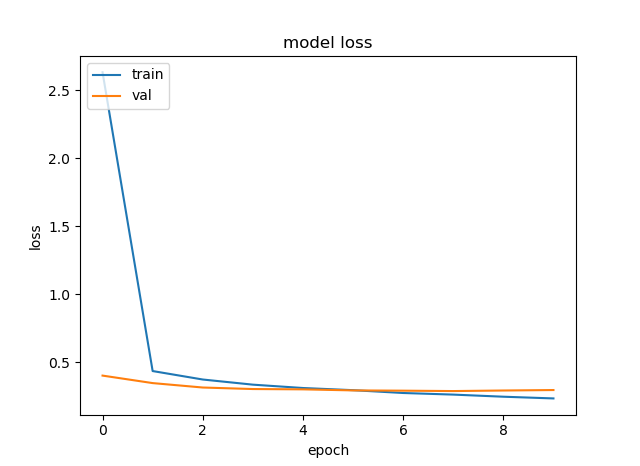
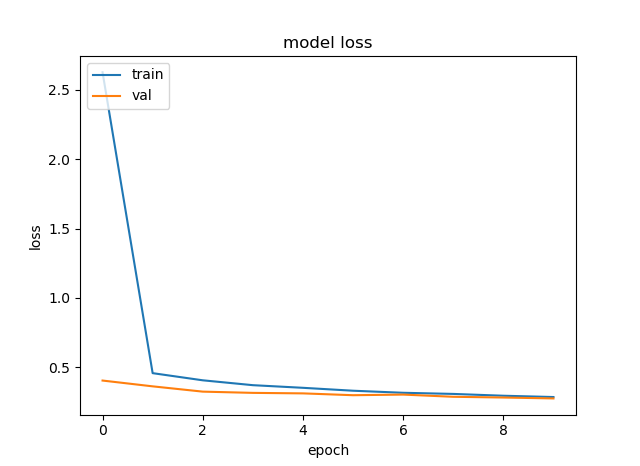
*Figure 1: Models Results and Accuracy*



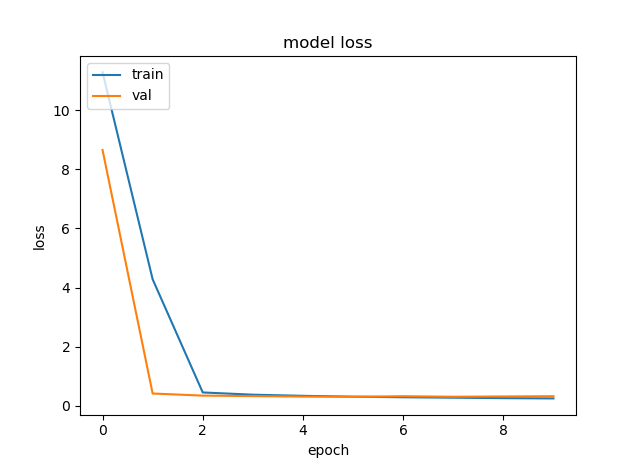
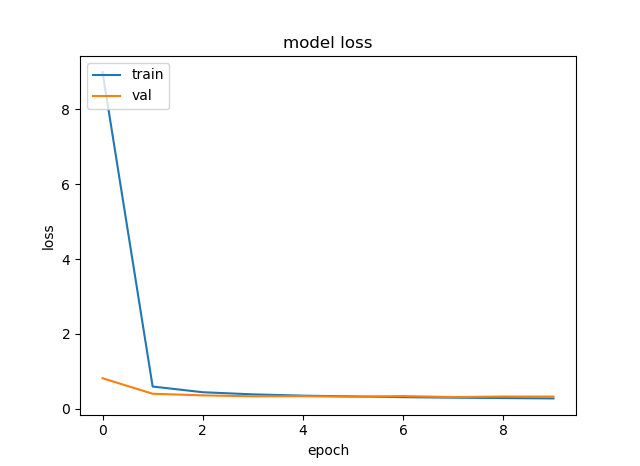
*Figure 2: Time to build each epoch per model in seconds*

### C:\Users\camer\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture1.png C:\Users\camer\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture2.png

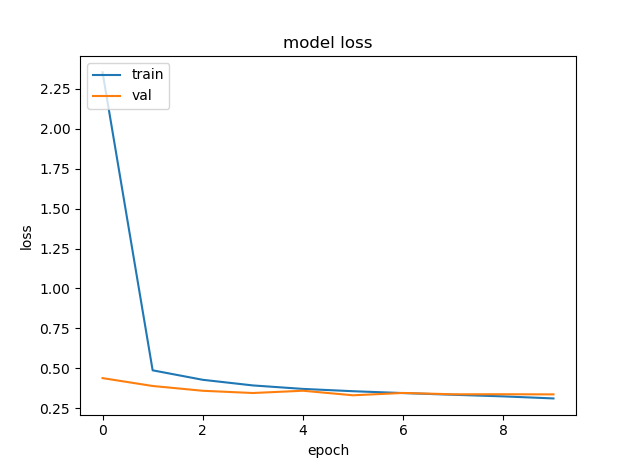
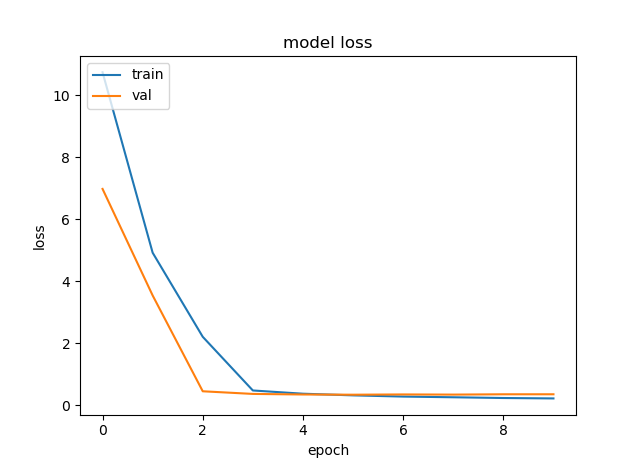
*Figure 3: MLP Model Loss Figure 4: CNN Model Loss*

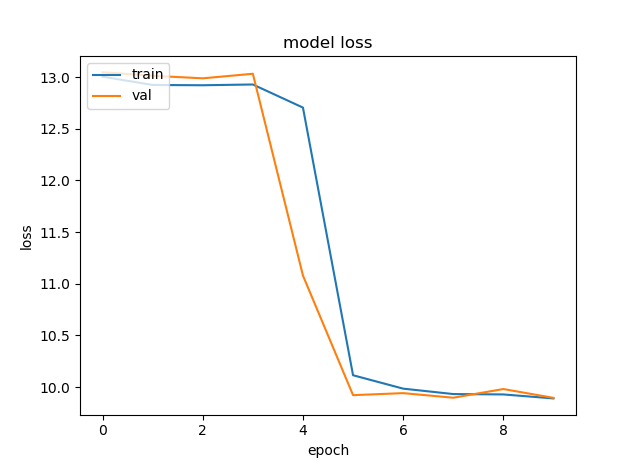
*Figure 5: CNN 2 convolutional layers Model Loss Figure 6: CNN 2 convolutional and pooling Model Loss*

*Figure 7: CNN 32 filter Model Loss Figure 8: CNN 5x5 filter Model Loss*

*Figure 9: CNN 7x7 filter Model Loss Figure 10: CNN no max pooling layer Model Loss*



*Figure 11: CNN no dropout layer Model Loss*

## Conclusion

The model with the best performance was CNN with two convolutional layers and two max pooling layers having an average accuracy of 89.27%. All models, apart from the CNN model with no dropout layer, lay in the range of 87.12% and 89.27%. This is a small range of 2.15% making them almost equal in performance.

The model that took the longest to make was CNN model with two convolutional layers and two max pooling layers. The model that had two convolutional layers and the model that had 32 filters was only slightly quicker to build. The original CNN model was on the quicker range to build with about 63 seconds per epoch and giving us an average accuracy of 88.91%, only .36% lower then the best performing model. This would probably be the best model to use at it is quicker then the longest to build and till performed very well.

The worst performing model was the CNN with no dropout layer. This had a horrible average accuracy of 27.06% and a build time of about 63 seconds per epoch. At that rate even the MLP model would be better to use at is had an average accuracy of 87.65% an about 2 seconds per epoch to build.

## Code

#imports

import matplotlib.pyplot as plt

import gzip

import os

import numpy as np

np.random.seed(1234)

#keras imports

from keras.datasets import mnist

from keras.models import Sequential

import keras

from keras.layers import Dense, Activation, convolutional, MaxPooling2D, Dropout, Flatten

from keras import optimizers

from keras.utils import np\_utils

from keras.utils.data\_utils import get\_file

from keras import backend as K

K.set\_image\_dim\_ordering('th')

np.random.seed(1234)

# Global Parameters

# model choice 'cnn' to select a ConvNet, anything else defaults to mlp

model = ''

# batch size and number of training epochs

batch\_size = 100

nb\_epoch = 10

# data input dimensions (adjust to use other data such as cifar which would be 32,32,3)

img\_rows, img\_cols,img\_channels = 28,28,1

# plots training and validation loss

def plot\_losses(history):

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'val'], loc='upper left')

plt.show()

#Loads the Fashion-MNIST dataset.

def load\_data():

"""Loads the Fashion-MNIST dataset.

# Returns

Tuple of Numpy arrays: `(x\_train, y\_train), (x\_test, y\_test)`.

"""

dirname = os.path.join('datasets', 'fashion-mnist')

base = 'http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/'

files = ['train-labels-idx1-ubyte.gz', 'train-images-idx3-ubyte.gz',

't10k-labels-idx1-ubyte.gz', 't10k-images-idx3-ubyte.gz']

paths = []

for file in files:

paths.append(get\_file(file, origin=base + file, cache\_subdir=dirname))

with gzip.open(paths[0], 'rb') as lbpath:

y\_train = np.frombuffer(lbpath.read(), np.uint8, offset=8)

with gzip.open(paths[1], 'rb') as imgpath:

x\_train = np.frombuffer(imgpath.read(), np.uint8,

offset=16).reshape(len(y\_train), 28, 28)

with gzip.open(paths[2], 'rb') as lbpath:

y\_test = np.frombuffer(lbpath.read(), np.uint8, offset=8)

with gzip.open(paths[3], 'rb') as imgpath:

x\_test = np.frombuffer(imgpath.read(), np.uint8,

offset=16).reshape(len(y\_test), 28, 28)

return (x\_train, y\_train), (x\_test, y\_test)

# mlp model

def mlp\_model():

model = Sequential()

model.add(Dense(128, input\_dim=(img\_rows\*img\_cols)))

model.add(Activation('relu'))

model.add(Dense(128))

model.add(Activation('relu'))

model.add(Dense(10))

model.add(Activation('softmax'))

return model

# cnn model

def cnn\_model():

model = Sequential()

model.add(convolutional.Conv2D(16, (3, 3), strides=1,

input\_shape=(img\_channels, img\_rows, img\_cols)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128))

model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(10))

model.add(Activation('softmax'))

return model

# load mnist data, format for cnn model

def load\_mnist\_cnn():

# load data

(x\_train, y\_train), (x\_test, y\_test) = load\_data()

# reshape data (for cnn)

x\_train = x\_train.reshape(x\_train.shape[0], img\_channels, img\_rows, img\_cols)

x\_test = x\_test.reshape(x\_test.shape[0], img\_channels, img\_rows, img\_cols)

# need categorical classes

y\_train = keras.utils.to\_categorical(y\_train, 10)

y\_test = keras.utils.to\_categorical(y\_test, 10)

return x\_train, y\_train, x\_test, y\_test

# load mnist data, format for mlp model

def load\_mnist\_mlp():

# load data

(x\_train, y\_train), (x\_test, y\_test) = load\_data()

# flatten data for mlp

x\_train = x\_train.reshape(x\_train.shape[0], img\_rows\*img\_cols\*img\_channels)

x\_test = x\_test.reshape(x\_test.shape[0], img\_rows\*img\_cols\*img\_channels)

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

x\_train /= 255

x\_test /= 255

# need categorical classes

y\_train = keras.utils.to\_categorical(y\_train, 10)

y\_test = keras.utils.to\_categorical(y\_test, 10)

return x\_train, y\_train, x\_test, y\_test

# controls which model is used based on model selection at top of code

if (model == 'cnn'):

x\_train, y\_train, x\_test, y\_test = load\_mnist\_cnn()

model = cnn\_model()

else:

x\_train, y\_train, x\_test, y\_test = load\_mnist\_mlp()

model = mlp\_model()

# define optimizer

adam = optimizers.Adam(lr=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=1e-08, decay=0.0)

# compile model

model.compile(loss='categorical\_crossentropy',

optimizer=adam,

metrics=['accuracy'])

# train model on training data

history = model.fit(x\_train, y\_train, batch\_size=batch\_size, epochs=nb\_epoch,

verbose=1, validation\_split=0.1)

# scores model on test data for chosen metric (accuracy)

score = model.evaluate(x\_test, y\_test, verbose=0)

# print accuracy

print(score[1])

# plots loss for training and validation data

plot\_losses(history)

# ends session, avoids potential error on program exit

K.clear\_session()