Deep Learning - Assignment 4

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

	Loss	Accuracy	Val Loss	Val Acc	Test Accuracy
MLP	0.2405	0.9097	0.3342	0.8784	0.8765
CNN	0.2596	0.9006	0.3132	0.8917	0.8891
CNN (2 conv)	0.2350	0.9098	0.3036	0.8956	0.8899
CNN (2 pools)	0.2869	0.8924	0.2796	0.8946	0.8927
CNN (32 filters	0.2482	0.9057	0.3135	0.8959	0.8914
CNN (5x5)	0.2891	0.8906	0.3261	0.8828	0.8786
CNN (7x7)	0.3168	0.8799	0.3444	0.8765	0.8712
CNN (no pool)	0.2149	0.9188	0.3670	0.8859	0.8791
CNN (no drop)	11.9006	0.2615	11.7964	0.2701	0.2706

Figure 1: Models Results and Accuracy

	Epochs										
	1	2	3	4	5	6	7	8	9	10	avg
MLP	2	2	2	2	2	2	2	2	2	2	2
CNN	64	63	63	63	64	63	63	63	63	63	63
CNN (2 conv)	127	127	128	130	130	130	130	130	130	130	129
CNN (2 pools)	130	132	132	130	131	130	131	131	132	130	131
CNN (32 filters	124	123	123	123	123	122	122	122	123	122	123
CNN (5x5)	60	59	59	59	59	59	59	59	59	59	59
CNN (7x7)	59	57	57	57	57	57	57	56	57	57	57
CNN (no pool)	59	58	58	58	58	58	58	58	58	58	58
CNN (no drop)	65	64	63	63	63	63	63	63	63	63	63

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

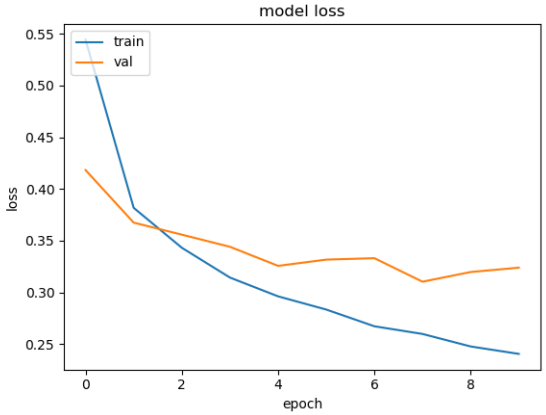

Figure 3: MLP Model Loss

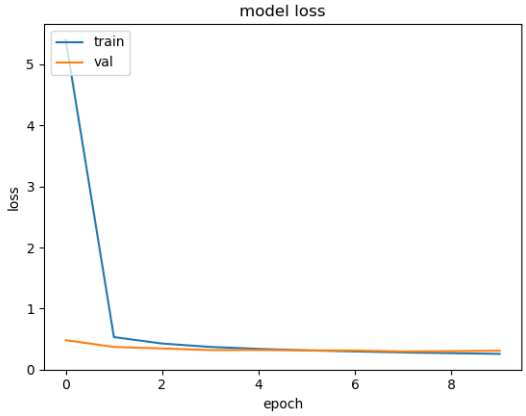

Figure 4: CNN Model Loss

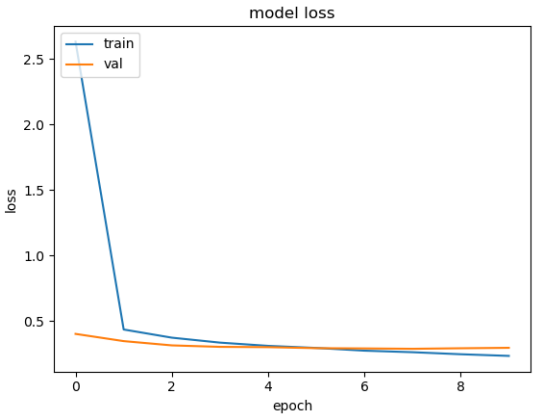

Figure 5: CNN 2 convolutional layers Model Loss

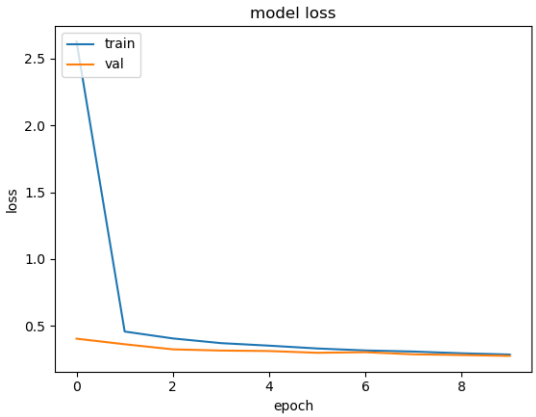

Figure 6: CNN 2 convolutional and pooling Model Loss

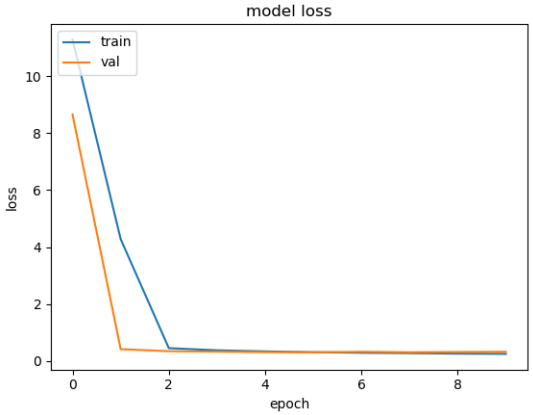

Figure 7: CNN 32 filter Model Loss

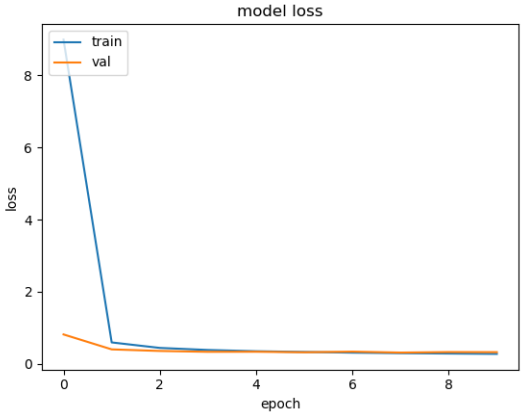

Figure 8: CNN 5x5 filter Model Loss

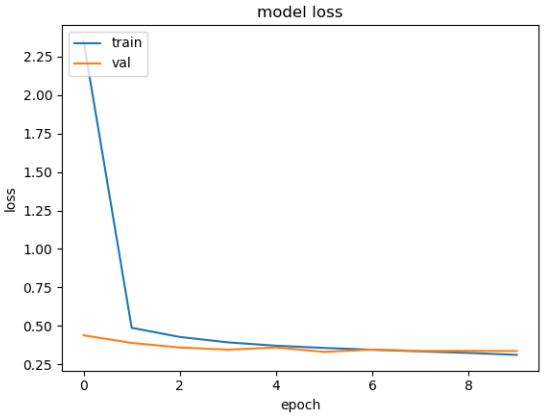

Figure 9: CNN 7x7 filter Model Loss

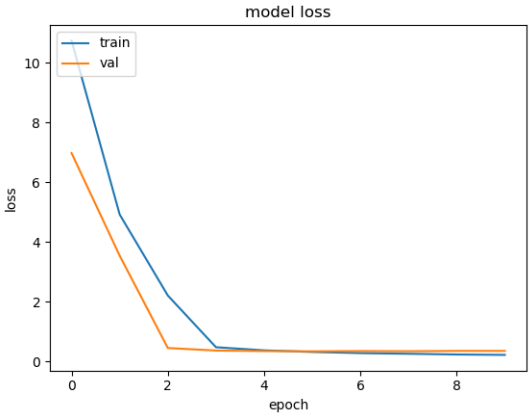

Figure 10: CNN no max pooling layer Model Loss

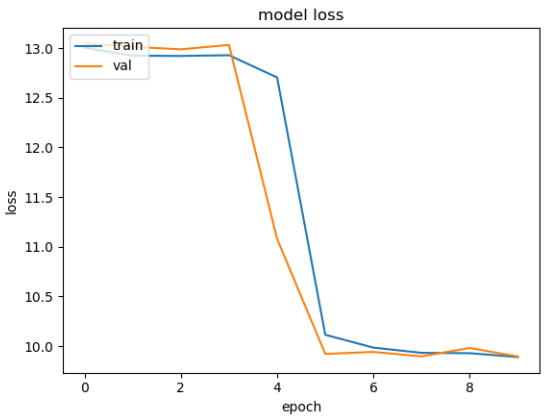

Figure 11: CNN no dropout layer Model Loss

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()
```