AIN433 Introduction to Computer Vision Lab. Practical 4 - Image Classification Using CNN & VGG16

Abdullah Palaz
21993016
Department of Computer Engineering
Hacettepe University
Ankara, Turkey
abdullahpalaz@hacettepe.edu.tr

Overview

The aim of this assignment is training CNN for image classification using TensorFlow and fine tuning a pretrained VGG16 on ImageNet data using transfer learning.

1 Dataset

The dataset is Indoor Scene Recognition Dataset from Computer Vision and Pattern Recognition Conference 2019, containing 15620 images in 67 categories. Due to computational and time limitations and assignment restrictions, a subset of ths dataset (15 category, each of which has more than or equal to 300 images) has been selected to be used in training/validation/testing. That is 7701 images of 15 category.

These categories are the following:

- kitchen: 734 images
- livingroom: 705 images
- bedroom: 662 images
- airport_inside: 608 images
- bar: 603 images
- subway: 539 images
- casino: 515 images
- restaurant: 513 images
- warehouse: 506 images
- inside_subway: 457 images
- bakery: 405 images
- pantry: 384 images
- bookstore: 380 images
- toystore: 347 images
- corridor: 343 images

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 100, 100, 16)      448
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 50, 50, 16)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 50, 50, 8)         1160
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 25, 25, 8)         0
_________________________________________________________________
flatten_5 (Flatten)          (None, 5000)              0
_________________________________________________________________
dense_10 (Dense)             (None, 32)                160032
_________________________________________________________________
dense_11 (Dense)             (None, 15)                495
=================================================================
Total params: 162,135
Trainable params: 162,135
Non-trainable params: 0
_________________________________________________________________
```

Figure 1: Topology of CNN without Dropout.

Train / test / validation splits has been done such that, 80% of the data is train data, 10% is test data and 10% is validation data. In numbers, there are 6160 images in training data, 770 images in test data and 771 images in validation data. Also, all images were resized to shape of (100, 100, 3) (channels last notation) and intensity values were scaled to 0 - 1. Note that images are not grayscale, instead, they are RGB.

2 CNN Part

There are two different topologies for CNN, one with dropout layers and one with no dropout layers.

2.1 CNN with no Dropouts

The topology of this CNN is shown in 1. Code snippet to create that topology can be seen in 1.

Each convolution layer is followed by a max pooling layer where pooling window size is 2x2. That choice was arbitrary. First convolution layer has 16 kernels, seconds has 8 kernels. Those choices were arbitrary too. Kernel sizes are 3x3, which is also an arbitrary choice. All activation functions except output layer's are ReLU, output layer's activation function is Softmax. Output layer's activation function choice was intentional, to make output values ranging from 0 to 1 (since they are probabilities), Softmax should be used. But other choices were arbitrary. Actually not arbitary but more like because of the convention. I mean, if everyone uses it that way, there must be some wisdom there I guess. Number of convolution layers (2) and dense layers (3, including output) are also results of arbitrary choices. Loss choice was "categorical cross entropy" and this was also due to convention. Everyone uses that way. Model does not yield that good results but that will be discussed later.

Listing 1: Code of CNN with no dropout

```
model = tf.keras.Sequential()
model.add(layers.InputLayer(input_shape=(HEIGHT, WIDTH, CHANNELS)))

model.add(layers.Conv2D(16, kernel_size=(3,3), padding='same', activation
    ='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))

model.add(layers.Conv2D(8, kernel_size=(3,3), padding='same', activation=
    'relu'))
```

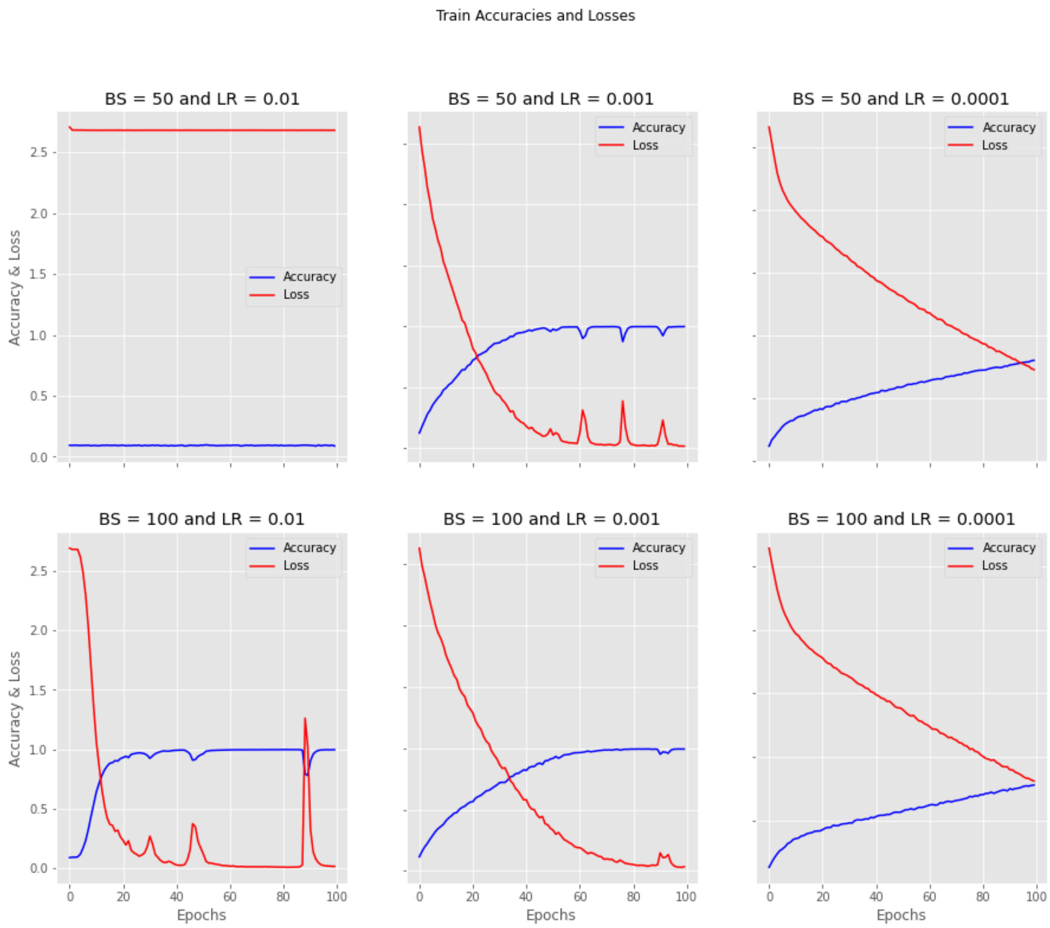

Figure 2: Train accuracies and losses of CNN models.

```
model.add(layers.MaxPool2D(pool_size=(2,2)))

model.add(layers.Flatten())
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(NUM_CLASSES, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=LEARNING_RATE), metrics=['accuracy'])
history = model.fit(X_train, y_train, batch_size=BATCH_SIZE, epochs=EPOCH_SIZE, validation_data = (X_validation, y_validation), shuffle=True)
```

Using epoch size as 100 (mendated), 2 different batch sizes (50 and 100) and 3 different learning rates (0.01, 0.001 and 0.0001) were tested in every possible combinations. That yields 6 different models, from a model with batch size of 50 and learning rate of 0.01 to a model with batch size of 100 and learning rate of 0.0001.

See the train accuracies and losses in 2 and validation accuracies and losses in 3 and test accuracies in 4.

See confusion matricies of the models in 5.

Confusion matrices says a lot about how model behaves. Looking at validation accuracy & loss graph, you can't tell if model just says random stuff, or as in current sample, just says everything is class "x". Model 0 thinks that every image belongs to a certain class. This happened probably due to learning

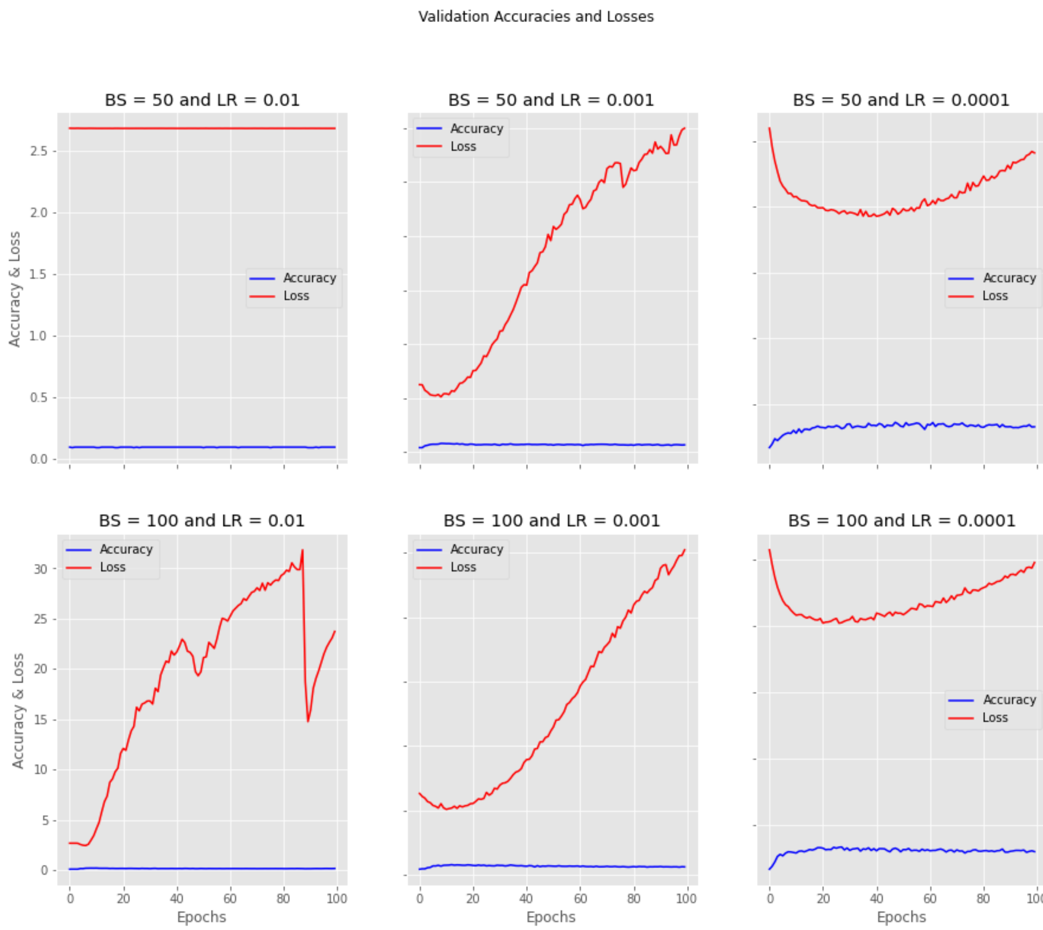

Figure 3: Validation accuracies and losses of CNN models.

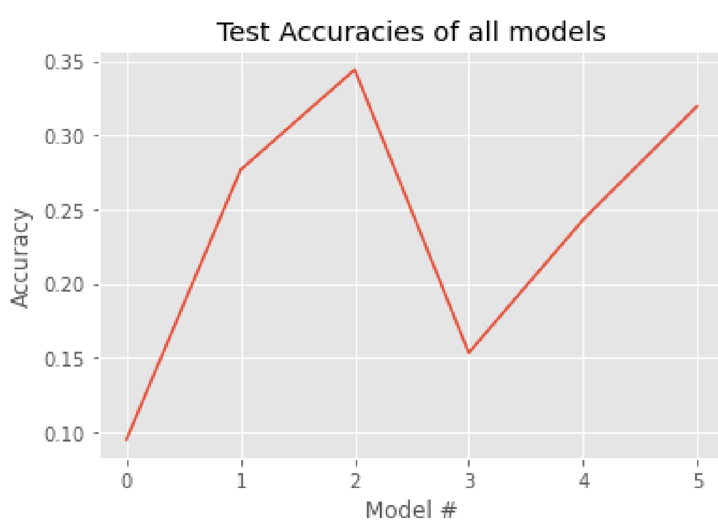

Figure 4: Test accuracies of CNN models.

```
Confusion matrix of model 0 with batch size of 50 and learning rate of 0.01

[[ 0  0  0  0  0  0  0  0 61  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 41  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 60  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 66  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 38  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 51  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 35  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 46  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 73  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 70  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 39  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 51  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 54  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 35  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 50  0  0  0  0  0  0]]
```

(a) Model 0

```
Confusion matrix of model 1 with batch size of 50 and learning rate of 0.001

[[17  3  6  0  1  0  0  8  2  3  3  0 11  1  6]
 [ 2  6  0  2  5  2  0  0  4  6  5  2  3  1  3]
 [ 1  4 19  0  3  8  1  0  3  5  3  9  3  1  0]
 [ 2  2  3 24  1  1  2  0 14 12  0  4  0  0  1]
 [ 2  1  5  0  4  0  0  2  2  4  8  2  1  5  2]
 [ 3  1 11  1  2 11  0  4  1  4  0  3  5  3  2]
 [ 0  0  0  2  1  1 16  1  5  1  4  1  2  0  1]
 [ 7  2  4  1  3  2  3 11  1  1  2  1  2  2  4]
 [ 3  4  5  6  1  0  1  0 29 11  3  6  3  0  1]
 [ 1  1  5 10  2  0  2  1 16 22  2  4  2  0  2]
 [ 1  2  1  0  2  0  0  0  3  3 16  6  0  0  5]
 [ 3  2  9  2  5  3  1  3  5  5  1  9  0  1  2]
 [ 9  3  4  0  3  3  0  2  5  2  2  2  9  3  7]
 [ 3  4  3  1  4  5  0  1  2  2  2  3  1  2  2]
 [ 6  1  2  0  4  3  0  0  3  2  4  0  4  3 18]]
```

(b) Model 1

```
Confusion matrix of model 2 with batch size of 50 and learning rate of 0.0001

[[30  2  2  1  1  0  0  5  1  3  2  0  6  1  7]
 [ 3 11  1  2  1  1  1  0  2  6  4  3  2  1  3]
 [ 3  2 16  2  2 12  0  0  3  9  2  7  2  0  0]
 [ 0  1  1 31  0  0  4  2  9 14  1  2  0  0  1]
 [ 3  1  5  0 11  2  1  2  0  3  2  3  1  3  1]
 [ 4  1  9  1  3 18  0  5  1  1  0  2  3  0  3]
 [ 2  2  1  4  0  2 20  0  0  3  0  0  1  0  0]
 [ 7  1  3  1  1  0  2 14  0  1  2  1  8  0  5]
 [ 1  0  1 11  0  0  5  1 27 14  1  5  1  0  6]
 [ 2  1  2 14  3  0  1  1 12 27  1  2  2  0  2]
 [ 4  3  0  0  3  0  1  0  3  5 15  2  0  1  2]
 [ 5  1  6  6  4  2  0  3  3  6  0 14  1  0  0]
 [13  1  4  2  4  3  0  3  0  2  3  2 10  1  6]
 [ 3  3  5  1  4  0  0  2  0  4  4  2  0  3  4]
 [ 9  0  2  2  3  1  0  1  3  4  1  2  2  2 18]]
```

(c) Model 2

```
Confusion matrix of model 3 with batch size of 100 and learning rate of 0.01

[[ 7  1  6  1  4  2  2  6  7  2  1  4 10  3  5]
 [ 4  1  2  1  4  5  1  2  3  1  1  4  5  5  2]
 [ 3  2 13  2  5 14  1  1  4  1  0  5  5  2  2]
 [ 3  3  1 10  1  0  5  2 15  7  6  6  3  1  3]
 [ 4  1  4  2  0  0  3  4  3  3  1  3  3  2  5]
 [ 6  1 10  1  0 11  0  4  3  1  0  4  9  0  1]
 [ 1  0  2  4  2  1  6  1  7  4  1  1  2  0  3]
 [ 5  1  3  1  1  6  0  8  2  3  2  4  7  0  3]
 [ 6  4  1  8  2  0  5  4 16 10  2  2  5  3  5]
 [ 4  2  6  9  4  2  1  2  6 11  4  8  8  2  1]
 [ 3  1  1  3  3  2  1  1  2  4  9  2  2  0  5]
 [ 4  1  6  3  2  8  0  1  4  8  2  3  5  2  2]
 [ 3  3  3  0  2  4  4  4  1  3  1  6  8  3  9]
 [ 2  3  3  4  5  1  0  2  3  1  1  2  3  2  3]
 [ 4  3  3  2  0  1  0  3  5  3  2  6  3  2 13]]
```

(d) Model 3

```
Confusion matrix of model 4 with batch size of 100 and learning rate of 0.001

[[22  2  2  1  0  2  2  4  2  0  3  3  8  3  7]
 [ 2  7  1  1  5  0  1  0  2  2  7  5  3  3  2]
 [ 3  4  9  2  5  7  0  2  0  6  0 10  7  2  3]
 [ 3  1  3 25  1  0  1  0 11 10  3  4  1  1  2]
 [ 4  7  5  1  3  0  0  0  0  3  3  3  5  2  2]
 [ 6  2  4  2  6  9  0  1  0  1  2  8  8  0  2]
 [ 1  1  1  4  0  0 15  0  4  3  2  0  2  0  2]
 [ 3  0  2  2  3  0  1 16  0  1  0  7  5  3  3]
 [ 3  2  3 18  0  1  2  1 24 10  3  3  2  0  1]
 [ 0  3  5 14  0  1  2  1 19 14  2  5  2  1  1]
 [ 4  5  2  2  1  0  0  0  3  0 15  1  2  2  2]
 [ 5  4  4  4  5  4  0  1  5  8  1  3  4  2  1]
 [10  1  4  1  2  4  2  2  3  2  4  2  6  6  5]
 [ 4  3  3  1  7  2  1  1  0  1  3  0  0  7  2]
 [ 9  2  2  1  1  1  3  0  2  2  3  1  8  3 12]]
```

(e) Model 4

```
Confusion matrix of model 5 with batch size of 100 and learning rate of 0.0001

[[24  0  3  2  3  2  1  4  3  2  1  1  6  4  5]
 [ 1  4  2  4  7  1  1  0  6  3  1  5  3  1  2]
 [ 1  1 12  0  5  9  2  1  5  6  1 12  3  2  0]
 [ 0  0  1 38  0  0  2  0  8  7  1  5  0  1  3]
 [ 1  2  2  1  9  4  0  1  2  1  1  7  1  2  4]
 [ 5  1  7  1  3 21  0  3  0  1  0  3  5  0  1]
 [ 1  0  1  3  0  0 22  0  2  1  1  1  1  0  2]
 [ 2  1  1  2  2  3  2 16  1  2  1  3  6  0  4]
 [ 1  1  1 17  0  1  6  1 23  9  0  5  4  1  3]
 [ 0  1  3 27  1  1  2  2 12 15  0  1  1  0  4]
 [ 2  2  2  1  1  1  2  0  8  1  9  3  2  1  4]
 [ 2  1  2  8  6 10  0  1  2  2  0 12  1  0  4]
 [ 4  1  0  3  5  3  1  2  3  2  1  4 16  1  8]
 [ 3  2  1  1  8  4  0  2  0  2  2  4  0  3  3]
 [ 5  0  0  4  2  4  1  1  3  2  2  1  3  0 22]]
```

(f) Model 5

Figure 5: Confusion Matricies of models without dropout.

rate. Adam optimizer's default learning rate is 0.001. Setting learning rate 10x of that (0.01) probably yield the result of oscillation. Making the batch size higher didn't change the situation as can been seen from model 3. Accuracy is still low in model 1 and model 4, and I have no idea why is that. This could be the moment where the model itself is the problem rather than parameters like batch size or learning rate. Because model 2 and model 5 also does not yield that good results. That being said, the dataset also seems challenging. Some images even hard for human interpretations. Lots of colours, shapes and 15 different categories makes this dataset challenging. Also, size of dataset may not be enough. So that should be considered either.

2.2 CNN with Dropouts

Also, dropout layer has been implemented to this topology because it was mandated in assignment instructions. Dropout layer added to end of every convolution layer and every dense layer except output layer. Resulting topology can be seen in 6.

Code snippet to create that topology can be seen in 2.

Listing 2: Code of CNN with dropout

```
model_ = tf.keras.Sequential()
model_.add(layers.InputLayer(input_shape=(HEIGHT, WIDTH, CHANNELS)))

model_.add(layers.Conv2D(16, kernel_size=(3,3), padding='same',
    activation='relu'))
model_.add(layers.MaxPool2D(pool_size=(2,2)))
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_22 (Conv2D)           (None, 100, 100, 16)      448
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 50, 50, 16)        0
_________________________________________________________________
dropout_15 (Dropout)         (None, 50, 50, 16)        0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 50, 50, 8)         1160
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 25, 25, 8)         0
_________________________________________________________________
dropout_16 (Dropout)         (None, 25, 25, 8)         0
_________________________________________________________________
flatten_11 (Flatten)         (None, 5000)              0
_________________________________________________________________
dense_22 (Dense)             (None, 32)                160032
_________________________________________________________________
dropout_17 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_23 (Dense)             (None, 15)                495
=================================================================
Total params: 162,135
Trainable params: 162,135
Non-trainable params: 0
_________________________________________________________________
```

Figure 6: Topology of CNN with Dropout.

```
model_.add(layers.Dropout(DROPOUT))

model_.add(layers.Conv2D(8, kernel_size=(3,3), padding='same', activation
    ='relu'))
model_.add(layers.MaxPool2D(pool_size=(2,2)))
model_.add(layers.Dropout(DROPOUT))

model_.add(layers.Flatten())
model_.add(layers.Dense(32, activation='relu'))
model_.add(layers.Dropout(DROPOUT))
model_.add(layers.Dense(NUM_CLASSES, activation='softmax'))

model_.compile(loss='categorical_crossentropy', optimizer=Adam(
    learning_rate=LEARNING_RATE), metrics=['accuracy'])

history_ = model_.fit(X_train, y_train, batch_size=BATCH_SIZE, epochs=
    EPOCH_SIZE, validation_data = (X_validation, y_validation), shuffle=
    True)
```

See the train / test / validation accuracy & loss graphs in 7, in 8 and in 9.

See their confusion matricies in 10.

Dropout was added after every layer part, because all of them are being trained and all of them may overfit. Using 4 different dropout rates (0.1, 0.25, 0.50 and 0.75) model has been tested. Test accuracies were same, and that is suprising. Human error was checked on that and seemed none, but this is always a possibility. Dropout didn't change much, actually different rates of dropout also didn't change much and this is probably due to lack of learning of the base model. I mean, add dropout or not, if model didn't learn anything, then what would that change?

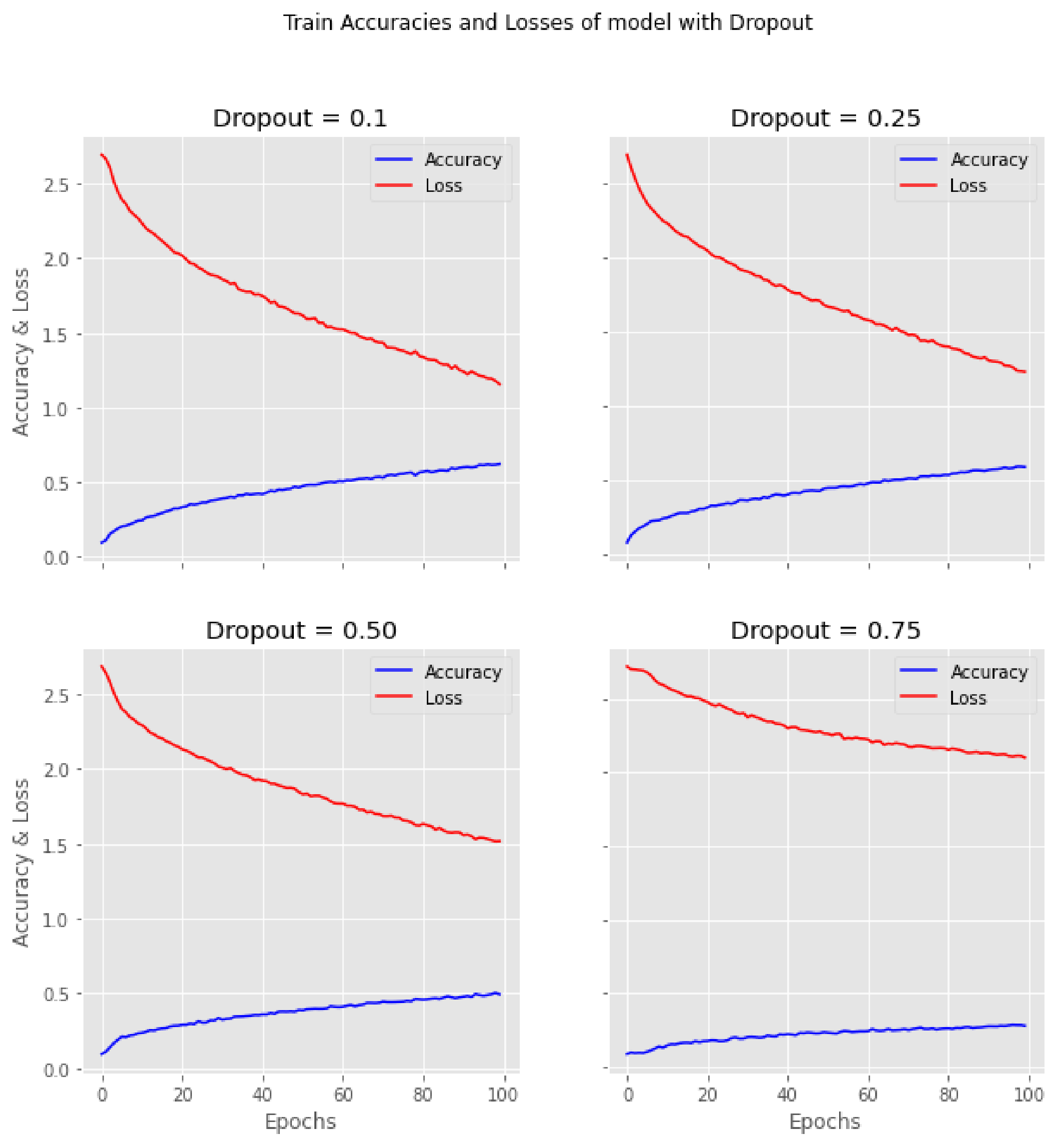

Figure 7: Train accuracies and losses of CNN models with dropout.

3 VGG Part

Fine tuning is a form of training in which, weights of a model are being fine tuned. These models usually were trained on large datasets such as ImageNet dataset. Taking the already trained model and tuning it for the dataset on hand, makes the learning faster than training a brand new neural network, and probably more robust to different kind of data samples. In general, last layers of the model (classification part of the models, see CNNs online for basic understanding) are being trained since all the thing that rest of the network do is extracting features. But in some cases, last few feature extraction layers also trained.

In this assignment, VGG16 that trained on ImageNet dataset has been used in two different configuration. In the first configuration, classification part of the VGG16 has been altered, such that last 3 Dense layers (classification part of the model) has 64, 32 and 15 neurons respectively (original VGG has different number of neurons in these layers) and unlike other layers, their weights has been initialized randomly. Layers other than classification part are freezed meaning that they will not be

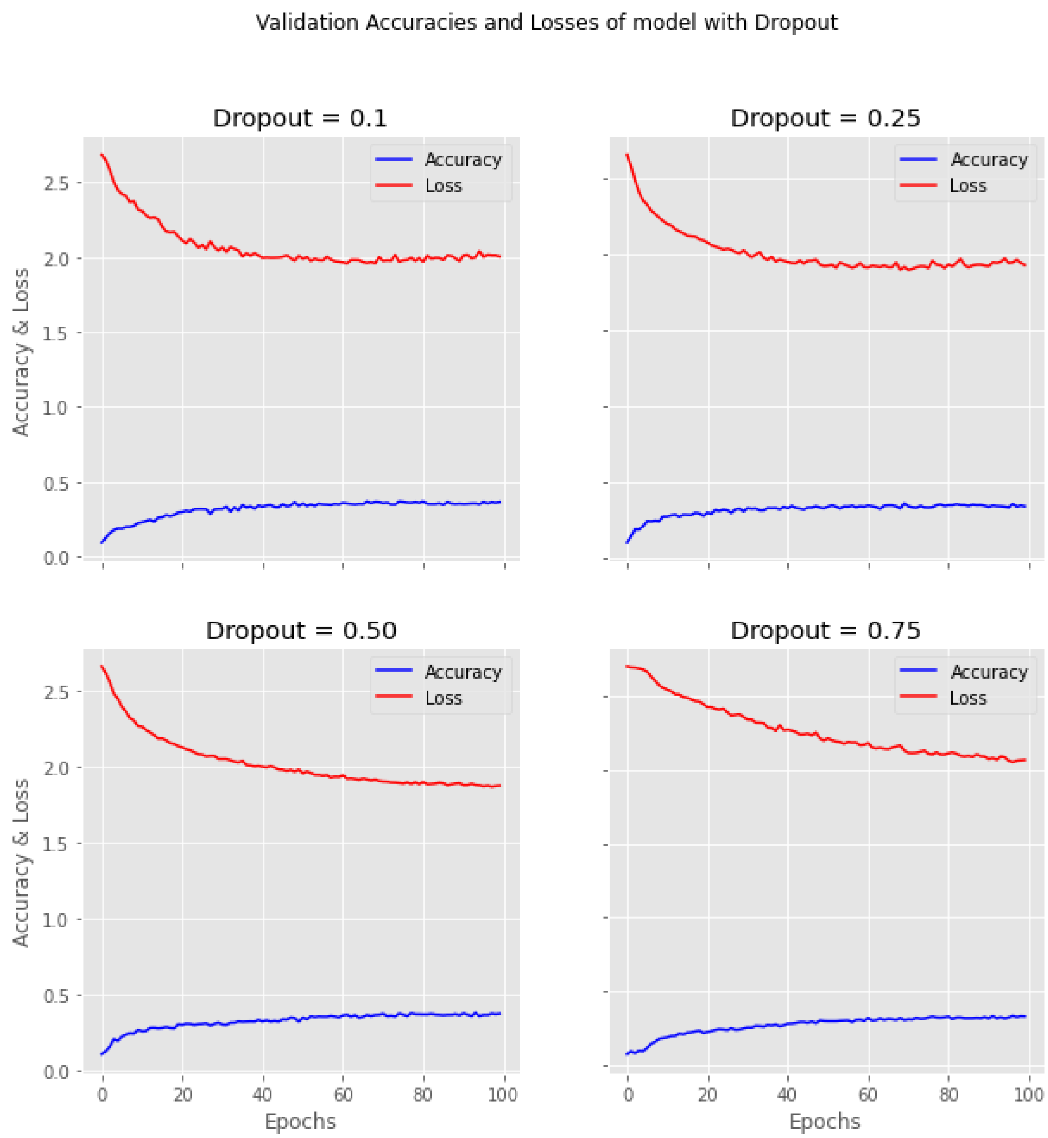

Figure 8: Validation accuracies and losses of CNN models with dropout.

trained. In the second configuration, on top of the first configuration, also last convolution layer is trainable however it's weights are not randomly initialized but initialized with VGG16 weights.

3.1 Configuration 1

Code to create the configuration 1 model can be seen in 3.

Listing 3: Code of VGG Configuration 1

```
vgg = VGG16(weights='imagenet', include_top=False, input_shape = (HEIGHT,
     WIDTH, CHANNELS))

for layer in vgg.layers:
    layer.trainable = False

model_vgg = tf.keras.Sequential([
    vgg,
    layers.Flatten(),
```

Figure 9: Test accuracies of CNN models with dropout.

```
Confusion matrix of model 0 with dropout of 0.1

[[24  0  3  2  3  2  1  4  3  2  1  1  6  4  5]
 [ 1  4  2  4  7  1  1  0  6  3  1  5  3  1  2]
 [ 1  1 12  0  5  9  2  1  5  6  1 12  3  2  0]
 [ 0  0  1 38  0  0  2  0  8  7  1  5  0  1  3]
 [ 1  2  2  1  9  4  0  1  2  1  1  7  1  2  4]
 [ 5  1  7  1  3 21  0  3  0  1  0  3  5  0  1]
 [ 1  0  1  3  0  0 22  0  2  1  1  1  1  0  2]
 [ 2  1  1  2  2  3  2 16  1  2  1  3  6  0  4]
 [ 1  1  1 17  0  1  6  1 23  9  0  5  4  1  3]
 [ 0  1  3 27  1  1  2  2 12 15  0  1  1  0  4]
 [ 2  2  2  1  1  1  2  0  8  1  9  3  2  1  4]
 [ 2  1  2  8  6 10  0  1  2  2  0 12  1  0  4]
 [ 4  1  0  3  5  3  1  2  3  2  1  4 16  1  8]
 [ 3  2  1  1  8  4  0  2  0  2  2  4  0  3  3]
 [ 5  0  0  4  2  4  1  1  3  2  2  1  3  0 22]]
```

(a) Model 0

```
Confusion matrix of model 1 with dropout of 0.25

[[24  0  3  2  3  2  1  4  3  2  1  1  6  4  5]
 [ 1  4  2  4  7  1  1  0  6  3  1  5  3  1  2]
 [ 1  1 12  0  5  9  2  1  5  6  1 12  3  2  0]
 [ 0  0  1 38  0  0  2  0  8  7  1  5  0  1  3]
 [ 1  2  2  1  9  4  0  1  2  1  1  7  1  2  4]
 [ 5  1  7  1  3 21  0  3  0  1  0  3  5  0  1]
 [ 1  0  1  3  0  0 22  0  2  1  1  1  1  0  2]
 [ 2  1  1  2  2  3  2 16  1  2  1  3  6  0  4]
 [ 1  1  1 17  0  1  6  1 23  9  0  5  4  1  3]
 [ 0  1  3 27  1  1  2  2 12 15  0  1  1  0  4]
 [ 2  2  2  1  1  1  2  0  8  1  9  3  2  1  4]
 [ 2  1  2  8  6 10  0  1  2  2  0 12  1  0  4]
 [ 4  1  0  3  5  3  1  2  3  2  1  4 16  1  8]
 [ 3  2  1  1  8  4  0  2  0  2  2  4  0  3  3]
 [ 5  0  0  4  2  4  1  1  3  2  2  1  3  0 22]]
```

(b) Model 1

```
Confusion matrix of model 2 with dropout of 0.5

[[24  0  3  2  3  2  1  4  3  2  1  1  6  4  5]
 [ 1  4  2  4  7  1  1  0  6  3  1  5  3  1  2]
 [ 1  1 12  0  5  9  2  1  5  6  1 12  3  2  0]
 [ 0  0  1 38  0  0  2  0  8  7  1  5  0  1  3]
 [ 1  2  2  1  9  4  0  1  2  1  1  7  1  2  4]
 [ 5  1  7  1  3 21  0  3  0  1  0  3  5  0  1]
 [ 1  0  1  3  0  0 22  0  2  1  1  1  1  0  2]
 [ 2  1  1  2  2  3  2 16  1  2  1  3  6  0  4]
 [ 1  1  1 17  0  1  6  1 23  9  0  5  4  1  3]
 [ 0  1  3 27  1  1  2  2 12 15  0  1  1  0  4]
 [ 2  2  2  1  1  1  2  0  8  1  9  3  2  1  4]
 [ 2  1  2  8  6 10  0  1  2  2  0 12  1  0  4]
 [ 4  1  0  3  5  3  1  2  3  2  1  4 16  1  8]
 [ 3  2  1  1  8  4  0  2  0  2  2  4  0  3  3]
 [ 5  0  0  4  2  4  1  1  3  2  2  1  3  0 22]]
```

(c) Model 2

```
Confusion matrix of model 3 with dropout of 0.75

[[24  0  3  2  3  2  1  4  3  2  1  1  6  4  5]
 [ 1  4  2  4  7  1  1  0  6  3  1  5  3  1  2]
 [ 1  1 12  0  5  9  2  1  5  6  1 12  3  2  0]
 [ 0  0  1 38  0  0  2  0  8  7  1  5  0  1  3]
 [ 1  2  2  1  9  4  0  1  2  1  1  7  1  2  4]
 [ 5  1  7  1  3 21  0  3  0  1  0  3  5  0  1]
 [ 1  0  1  3  0  0 22  0  2  1  1  1  1  0  2]
 [ 2  1  1  2  2  3  2 16  1  2  1  3  6  0  4]
 [ 1  1  1 17  0  1  6  1 23  9  0  5  4  1  3]
 [ 0  1  3 27  1  1  2  2 12 15  0  1  1  0  4]
 [ 2  2  2  1  1  1  2  0  8  1  9  3  2  1  4]
 [ 2  1  2  8  6 10  0  1  2  2  0 12  1  0  4]
 [ 4  1  0  3  5  3  1  2  3  2  1  4 16  1  8]
 [ 3  2  1  1  8  4  0  2  0  2  2  4  0  3  3]
 [ 5  0  0  4  2  4  1  1  3  2  2  1  3  0 22]]
```

(d) Model 3

Figure 10: Confusion Matricies of models without dropout.

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 100, 100, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 100, 100, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 100, 100, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 50, 50, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 50, 50, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 50, 50, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 25, 25, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 25, 25, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 25, 25, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 25, 25, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 12, 12, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 12, 12, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 12, 12, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 12, 12, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 6, 6, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 6, 6, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 6, 6, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 6, 6, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 3, 3, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 11: Topology of imported VGG.

```
    layers.Dense(64, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(NUM_CLASSES, activation="softmax")
])


model_vgg.compile(loss='categorical_crossentropy', optimizer="Adam",
    metrics=['accuracy'])

history_vgg = model_vgg.fit(X_train, y_train, batch_size=100, epochs=100,
     validation_data = (X_validation, y_validation), shuffle=True)
```

Also topology of imported VGG can be seen in 11 and topology of configuration 1 model can be seen in 12.

Number of neuron selection here highly affected by runtime because even with that number of neurons, training took 5 - 6 hours.

Train / validation / test accuracies and losses can be seen in 13, in 14 and in 15.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 3, 3, 512)         14714688
_________________________________________________________________
flatten_2 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_6 (Dense)              (None, 64)                294976
_________________________________________________________________
dense_7 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_8 (Dense)              (None, 15)                495
=================================================================
Total params: 15,012,239
Trainable params: 297,551
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 12: Topology of configuration 1 model.

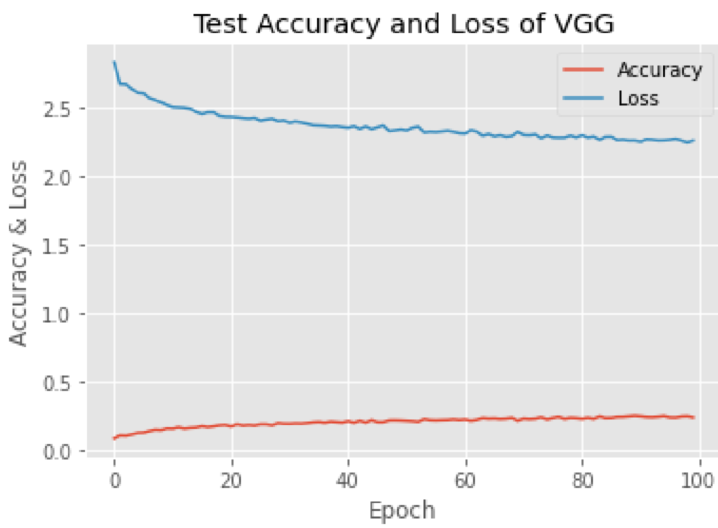

Figure 13: Train accuracy and loss of VGG Configuration 1.

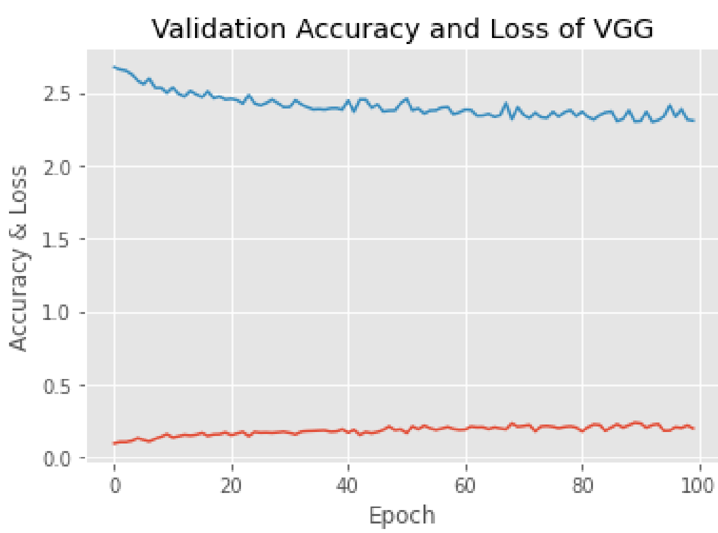

Figure 14: Validaion accuracy and loss of VGG Configuration 1.

```
In [26]: y_pred_vgg = model_vgg.predict(X_test)
         y_pred_classes_vgg = np.argmax(y_pred_vgg, axis=1)
         y_test_classes = np.argmax(y_test, axis=1)

         vgg_accuracy_fc = accuracy_score(y_test_classes, y_pred_classes_vgg)

         print("VGG that only Dense layers trained has accuracy of", vgg_accuracy_fc)

         VGG that only Dense layers trained has accuracy of 0.22467532467532467
```

Figure 15: Test accuracy of VGG Configuration 1.

```
Confusion matrix of VGG that only Dense layers trained
[[14  0  0  5  1  0  2  7  1  3  6  4  8  5  5]
 [ 7  4  0  1  2  0  0  1  1  3 12  2  0  6  2]
 [17  1  3  0  1  1  3  7  1  7  8  3  2  1  5]
 [ 2  0  0 34  0  0 10  1  1 13  1  1  2  0  1]
 [ 8  2  1  1  1  1  0  4  0  1 10  1  2  4  2]
 [14  0  1  2  0  0  0  7  0  3  8  6  2  4  4]
 [ 1  0  0  3  0  0 23  3  0  1  2  0  1  0  1]
 [ 5  0  0  0  0  1  3 21  0  0  5  3  2  0  6]
 [10  0  0 18  0  0  7  1  7 15  4  4  2  1  4]
 [ 5  1  1 18  0  1  3  4  5 15  4  5  3  0  5]
 [ 5  1  1  1  0  0  1  6  1  3 13  3  0  2  2]
 [ 9  0  0  5  0  1  0  3  1  8 13  9  1  0  1]
 [ 7  1  0  3  0  1  9  4  0  2  3  0 12  1 11]
 [ 6  3  0  1  2  1  0  2  0  1  8  0  0  6  5]
 [10  0  0  3  1  0  1  2  0  2 11  2  5  2 11]]
```

Figure 16: Confusion matrix of VGG Configuration 1.

Also, confusion matrix can be seen in 16.

Well, it didn't do a good job. That may be due to bad number of weights. Train accuracy is low too, meaning that no learning happened actually. Or this accuracy score might be a good accuracy score for this problem. I don't really know. Looking at the confusion matrix at 16, it can be seen that model has lots of incorrect predictions which make it seem like no learning happened during training.

3.2 Configuration 2

Code to create the configuration 1 model can be seen in 4.

Listing 4: Code of VGG Configuration 2

```
vgg = VGG16(weights='imagenet', include_top=False, input_shape = (HEIGHT,
    WIDTH, CHANNELS))

for layer in vgg.layers:
    layer.trainable = False

vgg.layers[-1].trainable = True
vgg.layers[-2].trainable = True

model_vgg_ = tf.keras.Sequential([
    vgg,
    layers.Flatten(),
    layers.Dense(64, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(NUM_CLASSES, activation="softmax")
])
model_vgg_.summary()

model_vgg.compile(loss='categorical_crossentropy', optimizer="Adam",
    metrics=['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 3, 3, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 64)                294976
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_2 (Dense)              (None, 15)                495
=================================================================
Total params: 15,012,239
Trainable params: 2,657,359
Non-trainable params: 12,354,880
_________________________________________________________________
```

Figure 17: Topology of configuration 2 model.

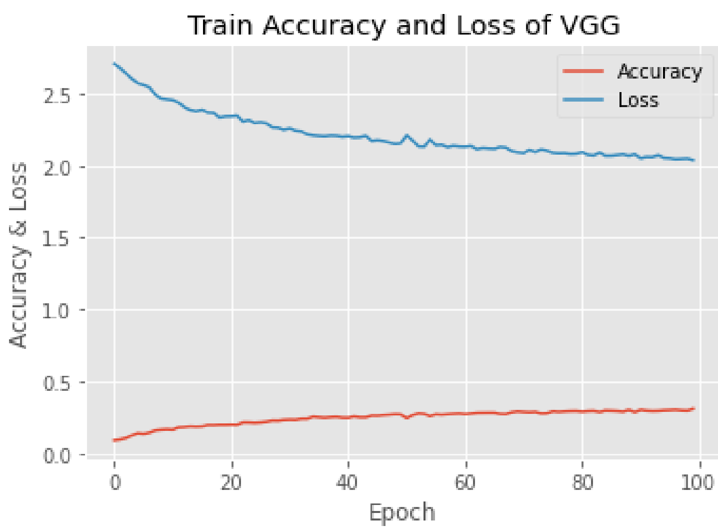

Figure 18: Train accuracy and loss of VGG Configuration 2.

```
history_vgg = model_vgg.fit(X_train, y_train, batch_size=100, epochs=100,
     validation_data = (X_validation, y_validation), shuffle=True)
```

Topology of configuration 1 model can be seen in 17.

Train / validation / test accuracies and losses can be seen in 18, in 19 and in 20.

Also, confusion matrix can be seen in 21.

Again, results are not promising but configuration 2 did better! So, a conclusion might be that, ImageNet classes are not aligned with the used dataset. Because when the last convolution layer trained (which is a feature extraction layer), the accuracy increased.

Being said these, one of the proposed models (model 2, batch size = 50 and learning rate = 0.0001) did better than fine tuning. Also training of the proposed models were way faster than fine tuning VGG16, in numbers training 6 different models took in total 43515 seconds and fine tuning VGG16 configuration 2 took 174807 seconds which is about 4x more. Also, altering CNN to get better results is much easier than fine tuninig VGG16 because of the both runtime, and that we have entire

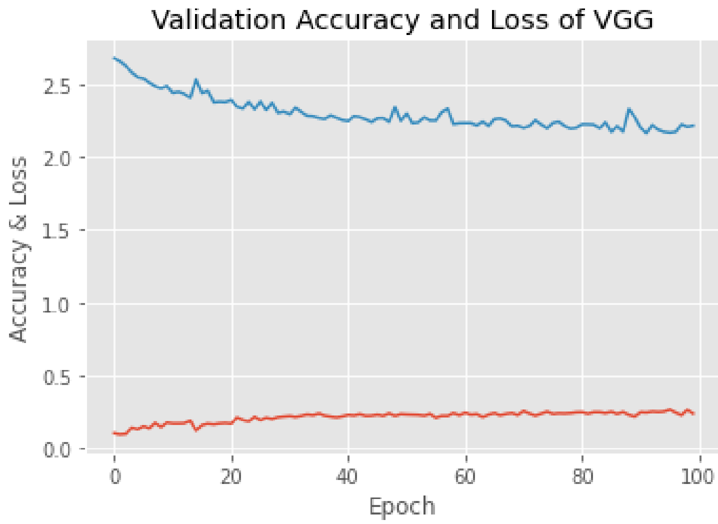

Figure 19: Validaion accuracy and loss of VGG Configuration 2.

```
In [25]: y_pred_vgg_ = model_vgg_.predict(X_test)
         y_pred_classes_vgg_ = np.argmax(y_pred_vgg_, axis=1)
         y_test_classes = np.argmax(y_test, axis=1)

         vgg_accuracy_fc_conf2 = accuracy_score(y_test_classes, y_pred_classes_vgg_)

         print("VGG that Dense layers & last Convolution layer trained has accuracy of", vgg_accuracy_fc_conf2)

         VGG that Dense layers & last Convolution layer trained has accuracy of 0.24675324675324675
```

Figure 20: Test accuracy of VGG Configuration 2.

```
Confusion matrix of VGG that Dense layers & last Convolution layer trained
[[ 0  0  2  6  4 10  0 14  2  1  0  7  8  2  5]
 [ 3  0  4  1  9  4  0  2  0  1  7  2  1  5  2]
 [ 2  0  2  0  8 17  3 11  2  1  2  6  2  2  2]
 [ 0  0  2 32  0  3 14  0  1  8  0  2  4  0  0]
 [ 0  0  2  0 11  8  0  6  0  0  2  4  2  3  0]
 [ 0  0  2  2  6 17  0 12  0  2  1  3  2  3  1]
 [ 1  0  0  5  0  1 21  3  0  0  0  1  3  0  0]
 [ 0  1  0  0  1  1  3 33  0  0  0  2  5  0  0]
 [ 1  0  6 22  1  2  7  4 10  8  1  9  0  0  2]
 [ 0  0  5 23  0  4  3  6  8 11  1  8  0  0  1]
 [ 1  0  3  0  6  2  1  7  2  1 11  1  0  2  2]
 [ 0  1  1  5  3 11  2  8  2  4  0 13  1  0  0]
 [ 0  1  1  1  2  7 12  6  2  1  0  0 17  0  4]
 [ 2  0  1  0  7  7  0  4  0  1  3  1  0  9  0]
 [ 1  1  5  1  7  9  3  5  0  2  1  2  8  2  3]]
```

Figure 21: Confusion matrix of VGG Configuration 2.

control of CNN while VGG16 comes with number of imported layers which is impossible (if not, hard) to alter.