Assignment 2: Build a CNN for image recognition.

Due Date: March 31, 11:59PM

Name: [Bhargavi Katta]

Introduction:

1. In this assignment, you will build Convolutional Neural Network to classify CIFAR-10 Images.
2. You can directly load dataset from many deep learning packages.
3. You can use any deep learning packages such as pytorch, keras or tensorflow for this assignment.

Requirements:

1. You need to load cifar 10 data and split the entire training dataset into training and validation.
2. You will implement a CNN model to classify cifar 10 images with provided structure.
3. You need to plot the training and validation accuracy or loss obtained from above step.
4. Then you can use tuned hyper-parameters to train using the entire training dataset.
5. You should report the testing accuracy using the model with complete data.
6. You may try to change the structure (e.g, add BN layer or dropout layer,...) and analyze your findings.

Google Colab

- If you do not have GPU, the training of a CNN can be slow. Google Colab is a good option.

Batch Normalization (BN)

Background:

- Batch Normalization is a technique to speed up training and help make the model more stable.

- In simple words, batch normalization is just another network layer that gets inserted between a hidden layer and the next hidden layer. Its job is to take the outputs from the first hidden layer and normalize them before passing them on as the input of the next hidden layer.

- For more detailed information, you may refer to the original paper: https://arxiv.org/pdf/1502.03167.pdf.

BN Algorithm:

- Input: Values of x over a mini-batch: $\mathbf{B}$ = $\{x_1, \ldots, x_m\}$;
- Output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$, γ, β are learnable parameters

Normalization of the Input:

$$\mu_{\mathbf{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_{\mathbf{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathbf{B}})^2$$

$$\hat{x}_i = \frac{x_i - \mu_{\mathbf{B}}}{\sqrt{\sigma_{\mathbf{B}}}^2 + \epsilon}$$

Re-scaling and Offsetting:

$$y_i = \gamma\hat{x}_i + \beta = BN_{\gamma,\beta}(x_i)$$

Advantages of BN:

1. Improves gradient flow through the network.
2. Allows use of saturating nonlinearities and higher learning rates.
3. Makes weights easier to initialize.
4. Act as a form of regularization and may reduce the need for dropout.

Implementation:

- The batch normalization layer has already been implemented in many packages. You may simply call the function to build the layer. For example: torch.nn.BatchNorm2d() using pytroch package, keras.layers.BatchNormalization() using keras package.
- The location of BN layer: Please make sure `BatchNormalization` is between a `Conv` / `Dense` layer and an `activation` layer.

1. Data preparation

1.1. Load data

In [1]:

```
import numpy as np
import tensorflow
import tensorflow.keras as keras
```

In [2]:

```
# Load Cifar-10 Data
# This is just an example, you may load dataset from other packages.
import tensorflow.keras as keras
import numpy as np

### If you can not load keras dataset, un-comment these two lines.
#import ssl
#ssl._create_default_https_context = ssl._create_unverified_context

(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

print('shape of x_train: ' + str(x_train.shape))
print('shape of y_train: ' + str(y_train.shape))
print('shape of x_test: ' + str(x_test.shape))
print('shape of y_test: ' + str(y_test.shape))
print('number of classes: ' + str(np.max(y_train) - np.min(y_train) + 1))
#np.max(y_train),np.min(y_train) + 1
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 3s 0us/step
shape of x_train: (50000, 32, 32, 3)
shape of y_train: (50000, 1)
shape of x_test: (10000, 32, 32, 3)
shape of y_test: (10000, 1)
number of classes: 10
```

In [3]:

```
y_train.shape[0]
```

Out[3]:

```
50000
```

In [4]:

```
import matplotlib.pyplot as plt

# Display the first image in the training set
plt.imshow(x_train[2])

# Set the plot title to the corresponding label
plt.title(y_train[2])

# Remove the x and y axis ticks
plt.xticks([])
plt.yticks([])

# Show the plot
plt.show()
```

```
/usr/local/lib/python3.9/dist-packages/matplotlib/text.py:1279: FutureWarning: elementwise comparison failed; r
eturning scalar instead, but in the future will perform elementwise comparison
  if s != self._text:
```

1.2. One-hot encode the labels (5 points)

In the input, a label is a scalar in $\{0, 1, \cdots, 9\}$. One-hot encode transform such a scalar to a 10-dim vector. E.g., a scalar `y_train[j]=3` is transformed to the vector `y_train_vec[j]=[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]` .

1. Implement a function `to_one_hot` that transforms an $n \times 1$ array to a $n \times 10$ matrix.

2. Apply the function to `y_train` and `y_test` .

In [5]:

```
def to_one_hot(y, num_class=10):
    y_vect=np.zeros((y.shape[0],num_class))
    for i in range(len(y)):
      y_vect[i][y[i]]=1
    return y_vect

y_train_vec = to_one_hot(y_train)
```

```
y_test_vec = to_one_hot(y_test)

print('Shape of y_train_vec: ' + str(y_train_vec.shape))
print('Shape of y_test_vec: ' + str(y_test_vec.shape))

print(y_train[0])
print(y_train_vec[0])
```

```
Shape of y_train_vec: (50000, 10)
Shape of y_test_vec: (10000, 10)
[6]
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
```

Remark: the outputs should be

- Shape of y_train_vec: (50000, 10)
- Shape of y_test_vec: (10000, 10)
- [6]
- [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]

1.3. Randomly partition the training set to training and validation sets (5 points)

Randomly partition the 50K training samples to 2 sets:

- a training set containing 40K samples: x_tr, y_tr
- a validation set containing 10K samples: x_val, y_val

In [6]:

```
# Generating 50k random numbers splitting first 40k as train and rest 10k as test by using rand indices
rand_index=np.random.permutation(50000)
train_index=rand_index[0:40000]
valid_index=rand_index[40000:50000]

x_tr=x_train[train_index,:]
y_tr=y_train_vec[train_index,:]

x_val=x_train[valid_index,:]
y_val=y_train_vec[valid_index,:]

print('Shape of x_tr: ' + str(x_tr.shape))
print('Shape of y_tr: ' + str(y_tr.shape))
print('Shape of x_val: ' + str(x_val.shape))
print('Shape of y_val: ' + str(y_val.shape))
```

```
Shape of x_tr: (40000, 32, 32, 3)
Shape of y_tr: (40000, 10)
Shape of x_val: (10000, 32, 32, 3)
Shape of y_val: (10000, 10)
```

2. Build a CNN and tune its hyper-parameters (50 points)

- Build a convolutional neural network model using the below structure:

- It should have a structure of: Conv - ReLU - Max Pool - ConV - ReLU - Max Pool - Dense - ReLU - Dense - Softmax

- In the graph 3@32x32 means the dimension of input image, 32@30x30 means it has 32 filters and the dimension now becomes 30x30 after the convolution.

- All convolutional layers (Conv) should have stride = 1 and no padding.
- Max Pooling has a pool size of 2 by 2.

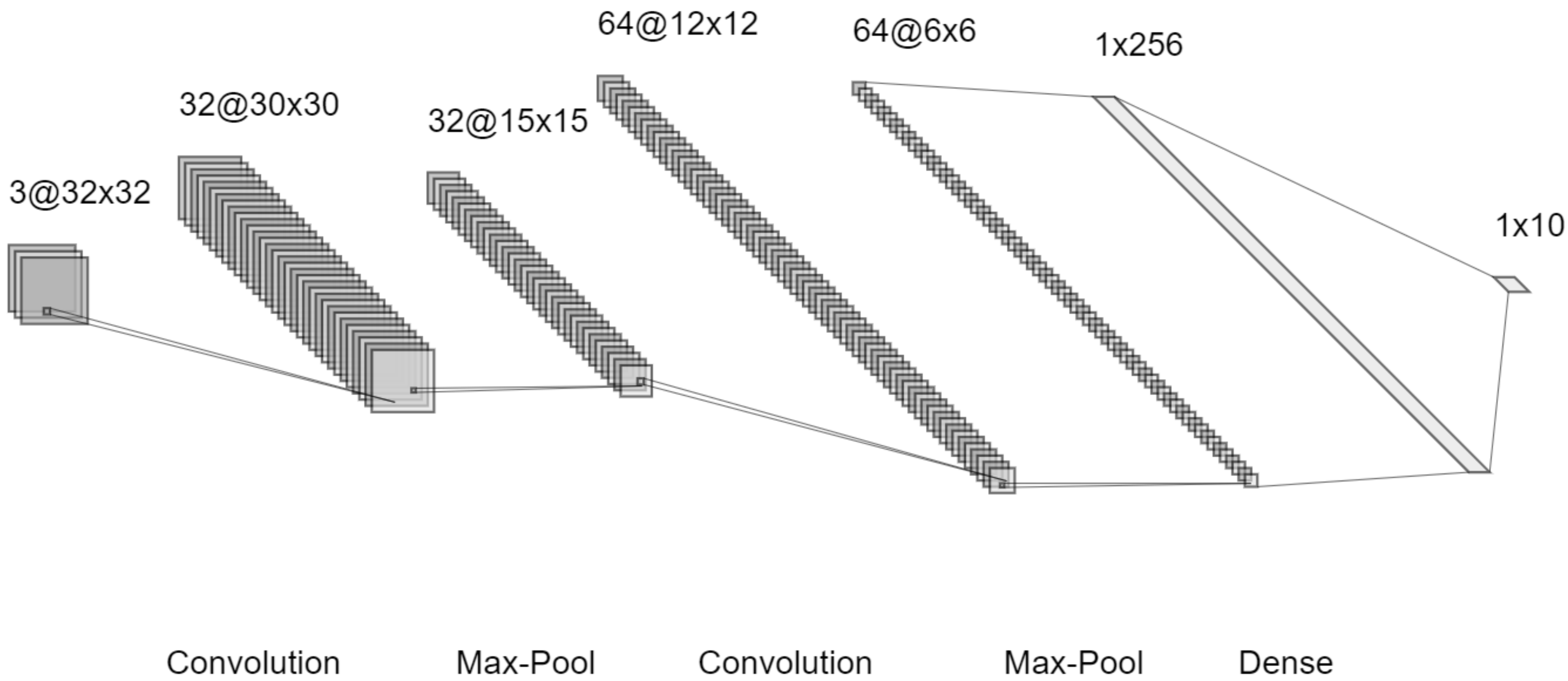
- You may use the validation data to tune the hyper-parameters (e.g., learning rate, and optimization algorithm)
- Do NOT use test data for hyper-parameter tuning!!!
- Try to achieve a validation accuracy as high as possible.

In [7]:

```
# Build the model
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout,Activation,BatchNormalization
from tensorflow.keras.models import Sequential
```

In [8]:

```
# padding='valid'means no. padding
# padding='same' means padding will be added to make the IP and OP shapes same
```



Basic CNN model with no BatchNormalization and No dropout Layers

In [9]:

```
model=Sequential()
model.add(Conv2D(32,(3,3),padding='valid',input_shape=(32,32,3)))
#model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D((2,2)))

model.add(Conv2D(64,(4,4),padding='valid'))
#model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D((2,2)))

model.add(Flatten())
model.add(Dense(256))
#model.add(BatchNormalization())
model.add(Activation("relu"))
```

```
model.add(Dense(10))
model.add(Activation('softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 activation (Activation)     (None, 30, 30, 32)        0

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 12, 12, 64)        32832

 activation_1 (Activation)   (None, 12, 12, 64)        0

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 256)               590080

 activation_2 (Activation)   (None, 256)               0

 dense_1 (Dense)             (None, 10)                2570

 activation_3 (Activation)   (None, 10)                0

=================================================================
Total params: 626,378
Trainable params: 626,378
Non-trainable params: 0
_________________________________________________________________
```

In [10]:

```
# Define model optimizer and loss function
from keras import optimizers

learning_rate = 1e-4 # tuned one
model.compile(loss='categorical_crossentropy',
```

```
              optimizer=optimizers.RMSprop(learning_rate=learning_rate),
              metrics=['acc'])
```

In [11]:

```
# Train the model and store model parameters/loss values

history = model.fit(x_tr,y_tr, epochs=50, validation_data=(x_val,y_val))
```

```
Epoch 1/50
1250/1250 [==============================] - 90s 71ms/step - loss: 2.8505 - acc: 0.3634 - val_loss: 1.5208 - va
l_acc: 0.4676
Epoch 2/50
1250/1250 [==============================] - 85s 68ms/step - loss: 1.3436 - acc: 0.5336 - val_loss: 1.3217 - va
l_acc: 0.5488
Epoch 3/50
1250/1250 [==============================] - 84s 67ms/step - loss: 1.1326 - acc: 0.6094 - val_loss: 1.2480 - va
l_acc: 0.5787
Epoch 4/50
1250/1250 [==============================] - 85s 68ms/step - loss: 0.9783 - acc: 0.6658 - val_loss: 1.2190 - va
l_acc: 0.5938
Epoch 5/50
1250/1250 [==============================] - 90s 72ms/step - loss: 0.8603 - acc: 0.7038 - val_loss: 1.1427 - va
l_acc: 0.6241
Epoch 6/50
1250/1250 [==============================] - 84s 67ms/step - loss: 0.7509 - acc: 0.7465 - val_loss: 1.2704 - va
l_acc: 0.5986
Epoch 7/50
1250/1250 [==============================] - 89s 71ms/step - loss: 0.6572 - acc: 0.7774 - val_loss: 1.1943 - va
l_acc: 0.6315
Epoch 8/50
1250/1250 [==============================] - 83s 66ms/step - loss: 0.5766 - acc: 0.8070 - val_loss: 1.2414 - va
l_acc: 0.6342
Epoch 9/50
1250/1250 [==============================] - 87s 70ms/step - loss: 0.4976 - acc: 0.8342 - val_loss: 1.2521 - va
l_acc: 0.6330
Epoch 10/50
1250/1250 [==============================] - 84s 68ms/step - loss: 0.4283 - acc: 0.8556 - val_loss: 1.3258 - va
l_acc: 0.6363
Epoch 11/50
1250/1250 [==============================] - 85s 68ms/step - loss: 0.3679 - acc: 0.8782 - val_loss: 1.2845 - va
l_acc: 0.6508
Epoch 12/50
1250/1250 [==============================] - 85s 68ms/step - loss: 0.3135 - acc: 0.8969 - val_loss: 1.4250 - va
l_acc: 0.6462
Epoch 13/50
```

```
1250/1250 [==============================] - 86s 69ms/step - loss: 0.2680 - acc: 0.9135 - val_loss: 1.4972 - va
l_acc: 0.6462
Epoch 14/50
1250/1250 [==============================] - 85s 68ms/step - loss: 0.2256 - acc: 0.9280 - val_loss: 1.5378 - va
l_acc: 0.6496
Epoch 15/50
1250/1250 [==============================] - 89s 71ms/step - loss: 0.1908 - acc: 0.9395 - val_loss: 1.6922 - va
l_acc: 0.6427
Epoch 16/50
1250/1250 [==============================] - 87s 69ms/step - loss: 0.1640 - acc: 0.9480 - val_loss: 1.6249 - va
l_acc: 0.6542
Epoch 17/50
1250/1250 [==============================] - 83s 67ms/step - loss: 0.1384 - acc: 0.9560 - val_loss: 1.8767 - va
l_acc: 0.6462
Epoch 18/50
1250/1250 [==============================] - 85s 68ms/step - loss: 0.1188 - acc: 0.9636 - val_loss: 1.8716 - va
l_acc: 0.6402
Epoch 19/50
1250/1250 [==============================] - 85s 68ms/step - loss: 0.1033 - acc: 0.9685 - val_loss: 1.9399 - va
l_acc: 0.6528
Epoch 20/50
1250/1250 [==============================] - 84s 67ms/step - loss: 0.0862 - acc: 0.9739 - val_loss: 2.1023 - va
l_acc: 0.6505
Epoch 21/50
1250/1250 [==============================] - 85s 68ms/step - loss: 0.0790 - acc: 0.9758 - val_loss: 2.2188 - va
l_acc: 0.6522
Epoch 22/50
1250/1250 [==============================] - 89s 71ms/step - loss: 0.0682 - acc: 0.9786 - val_loss: 2.2241 - va
l_acc: 0.6492
Epoch 23/50
1250/1250 [==============================] - 91s 72ms/step - loss: 0.0622 - acc: 0.9810 - val_loss: 2.1600 - va
l_acc: 0.6564
Epoch 24/50
1250/1250 [==============================] - 84s 67ms/step - loss: 0.0566 - acc: 0.9824 - val_loss: 2.3713 - va
l_acc: 0.6541
Epoch 25/50
1250/1250 [==============================] - 91s 73ms/step - loss: 0.0537 - acc: 0.9831 - val_loss: 2.5159 - va
l_acc: 0.6463
Epoch 26/50
1250/1250 [==============================] - 84s 67ms/step - loss: 0.0464 - acc: 0.9855 - val_loss: 2.5767 - va
l_acc: 0.6558
Epoch 27/50
1250/1250 [==============================] - 87s 70ms/step - loss: 0.0467 - acc: 0.9857 - val_loss: 2.6550 - va
l_acc: 0.6487
Epoch 28/50
```

```
1250/1250 [==============================] - 83s 67ms/step - loss: 0.0410 - acc: 0.9873 - val_loss: 2.6772 - va
l_acc: 0.6475
Epoch 29/50
1250/1250 [==============================] - 90s 72ms/step - loss: 0.0409 - acc: 0.9868 - val_loss: 2.6434 - va
l_acc: 0.6555
Epoch 30/50
1250/1250 [==============================] - 87s 70ms/step - loss: 0.0389 - acc: 0.9877 - val_loss: 3.0318 - va
l_acc: 0.6543
Epoch 31/50
1250/1250 [==============================] - 84s 67ms/step - loss: 0.0379 - acc: 0.9885 - val_loss: 2.9329 - va
l_acc: 0.6578
Epoch 32/50
1250/1250 [==============================] - 86s 69ms/step - loss: 0.0377 - acc: 0.9881 - val_loss: 2.8687 - va
l_acc: 0.6541
Epoch 33/50
1250/1250 [==============================] - 85s 68ms/step - loss: 0.0360 - acc: 0.9884 - val_loss: 3.0531 - va
l_acc: 0.6580
Epoch 34/50
1250/1250 [==============================] - 86s 69ms/step - loss: 0.0350 - acc: 0.9887 - val_loss: 3.1689 - va
l_acc: 0.6494
Epoch 35/50
1250/1250 [==============================] - 84s 67ms/step - loss: 0.0352 - acc: 0.9896 - val_loss: 3.1758 - va
l_acc: 0.6514
Epoch 36/50
1250/1250 [==============================] - 90s 72ms/step - loss: 0.0344 - acc: 0.9890 - val_loss: 3.4328 - va
l_acc: 0.6364
Epoch 37/50
1250/1250 [==============================] - 85s 68ms/step - loss: 0.0325 - acc: 0.9905 - val_loss: 3.2936 - va
l_acc: 0.6391
Epoch 38/50
1250/1250 [==============================] - 85s 68ms/step - loss: 0.0378 - acc: 0.9889 - val_loss: 3.4709 - va
l_acc: 0.6549
Epoch 39/50
1250/1250 [==============================] - 91s 73ms/step - loss: 0.0361 - acc: 0.9891 - val_loss: 3.2520 - va
l_acc: 0.6560
Epoch 40/50
1250/1250 [==============================] - 84s 67ms/step - loss: 0.0376 - acc: 0.9892 - val_loss: 3.7481 - va
l_acc: 0.6427
Epoch 41/50
1250/1250 [==============================] - 85s 68ms/step - loss: 0.0343 - acc: 0.9891 - val_loss: 3.0962 - va
l_acc: 0.6557
Epoch 42/50
1250/1250 [==============================] - 85s 68ms/step - loss: 0.0339 - acc: 0.9892 - val_loss: 3.3887 - va
l_acc: 0.6570
Epoch 43/50
```

```
1250/1250 [==============================] - 84s 67ms/step - loss: 0.0319 - acc: 0.9909 - val_loss: 3.8516 - va
l_acc: 0.6420
Epoch 44/50
1250/1250 [==============================] - 91s 73ms/step - loss: 0.0322 - acc: 0.9904 - val_loss: 3.6556 - va
l_acc: 0.6637
Epoch 45/50
1250/1250 [==============================] - 85s 68ms/step - loss: 0.0325 - acc: 0.9898 - val_loss: 3.4109 - va
l_acc: 0.6510
Epoch 46/50
1250/1250 [==============================] - 89s 71ms/step - loss: 0.0364 - acc: 0.9897 - val_loss: 3.6340 - va
l_acc: 0.6524
Epoch 47/50
1250/1250 [==============================] - 90s 72ms/step - loss: 0.0340 - acc: 0.9904 - val_loss: 3.5560 - va
l_acc: 0.6616
Epoch 48/50
1250/1250 [==============================] - 84s 67ms/step - loss: 0.0320 - acc: 0.9911 - val_loss: 3.6995 - va
l_acc: 0.6473
Epoch 49/50
1250/1250 [==============================] - 87s 69ms/step - loss: 0.0318 - acc: 0.9907 - val_loss: 3.7156 - va
l_acc: 0.6509
Epoch 50/50
1250/1250 [==============================] - 84s 67ms/step - loss: 0.0297 - acc: 0.9909 - val_loss: 3.9056 - va
l_acc: 0.6565
```

3. Plot the training and validation loss curve versus epochs. (5 points)

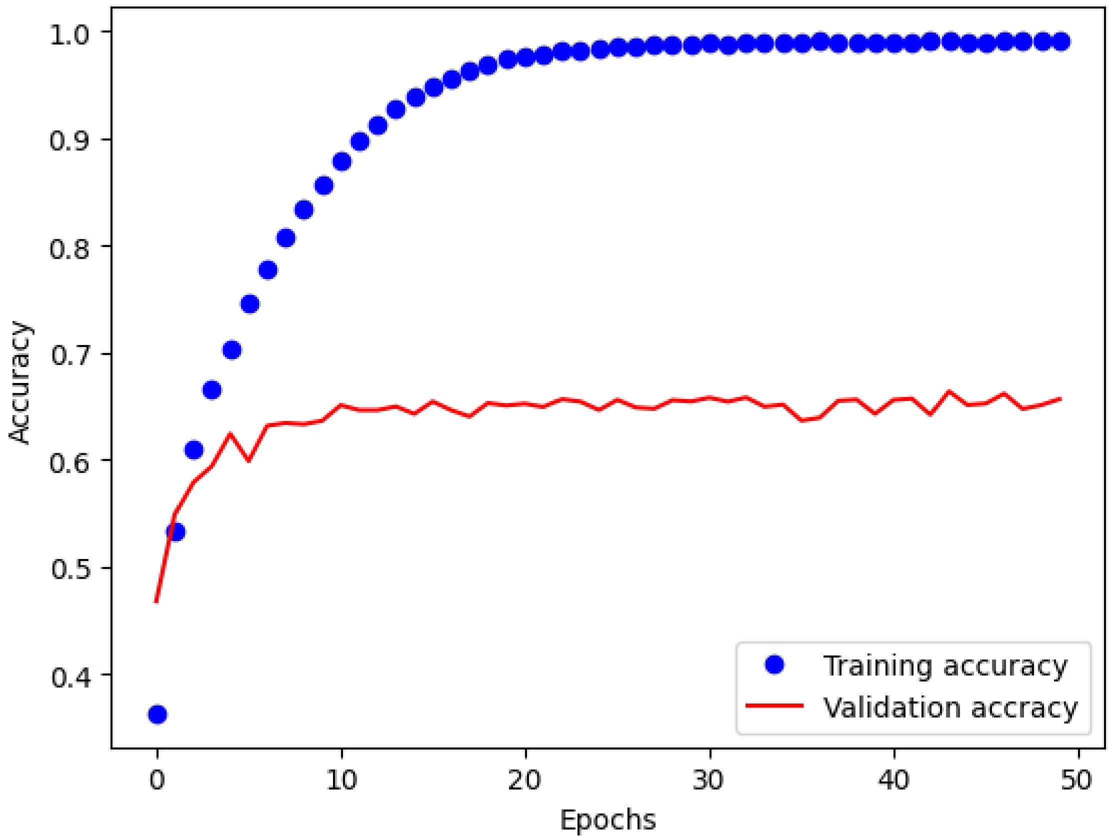
In [12]:

```python
# Plot the loss curve
import matplotlib.pyplot as plt
%matplotlib inline

acc = history.history['acc']
val_acc = history.history['val_acc']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [13]:

```
val_acc = history.history['val_acc']
train_acc=history.history['acc']
avg_val_acc = sum(val_acc)/len(val_acc)
avg_train_acc=sum(train_acc)/len(train_acc)
print("Average validation accuracy:", avg_val_acc)
print("Average train accuracy:", avg_train_acc)
```

```
Average validation accuracy: 0.6396780049800873
Average train accuracy: 0.9177804964780808
```

4. Train (again) and evaluate the model (5 points)

- To this end, you have found the "best" hyper-parameters.
- Now, fix the hyper-parameters and train the network on the entire training set (all the 50K training samples)
- Evaluate your model on the test set.

Train the model on the entire training set

Why? Previously, you used 40K samples for training; you wasted 10K samples for the sake of hyper-parameter tuning. Now you already know the hyper-parameters, so why not using all the 50K samples for training?

In [14]:

```
#<Compile your model again (using the same hyper-parameters you tuned above)>
learning_rate = 1e-4

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(learning_rate=learning_rate),
              metrics=['acc'])
```

In [15]:

```
#<Train your model on the entire training set (50K samples)>
history = model.fit(x_train,y_train_vec, epochs=25)
```

```
Epoch 1/25
1563/1563 [==============================] - 100s 64ms/step - loss: 0.6591 - acc: 0.8880
Epoch 2/25
1563/1563 [==============================] - 98s 63ms/step - loss: 0.4450 - acc: 0.9078
Epoch 3/25
1563/1563 [==============================] - 100s 64ms/step - loss: 0.3358 - acc: 0.9246
Epoch 4/25
1563/1563 [==============================] - 98s 63ms/step - loss: 0.2654 - acc: 0.9347
Epoch 5/25
1563/1563 [==============================] - 99s 63ms/step - loss: 0.2164 - acc: 0.9429
Epoch 6/25
1563/1563 [==============================] - 99s 64ms/step - loss: 0.1728 - acc: 0.9517
Epoch 7/25
1563/1563 [==============================] - 99s 63ms/step - loss: 0.1496 - acc: 0.9567
Epoch 8/25
1563/1563 [==============================] - 100s 64ms/step - loss: 0.1335 - acc: 0.9602
Epoch 9/25
1563/1563 [==============================] - 98s 62ms/step - loss: 0.1195 - acc: 0.9643
Epoch 10/25
1563/1563 [==============================] - 100s 64ms/step - loss: 0.1086 - acc: 0.9673
Epoch 11/25
1563/1563 [==============================] - 99s 63ms/step - loss: 0.0975 - acc: 0.9707
Epoch 12/25
1563/1563 [==============================] - 100s 64ms/step - loss: 0.0934 - acc: 0.9709
Epoch 13/25
1563/1563 [==============================] - 98s 63ms/step - loss: 0.0871 - acc: 0.9744
```

```
Epoch 14/25
1563/1563 [==============================] - 99s 63ms/step - loss: 0.0860 - acc: 0.9746
Epoch 15/25
1563/1563 [==============================] - 100s 64ms/step - loss: 0.0807 - acc: 0.9760
Epoch 16/25
1563/1563 [==============================] - 99s 63ms/step - loss: 0.0790 - acc: 0.9776
Epoch 17/25
1563/1563 [==============================] - 100s 64ms/step - loss: 0.0773 - acc: 0.9773
Epoch 18/25
1563/1563 [==============================] - 99s 64ms/step - loss: 0.0776 - acc: 0.9782
Epoch 19/25
1563/1563 [==============================] - 100s 64ms/step - loss: 0.0710 - acc: 0.9786
Epoch 20/25
1563/1563 [==============================] - 99s 63ms/step - loss: 0.0732 - acc: 0.9793
Epoch 21/25
1563/1563 [==============================] - 101s 64ms/step - loss: 0.0677 - acc: 0.9789
Epoch 22/25
1563/1563 [==============================] - 100s 64ms/step - loss: 0.0692 - acc: 0.9802
Epoch 23/25
1563/1563 [==============================] - 102s 65ms/step - loss: 0.0662 - acc: 0.9802
Epoch 24/25
1563/1563 [==============================] - 100s 64ms/step - loss: 0.0676 - acc: 0.9799
Epoch 25/25
1563/1563 [==============================] - 102s 65ms/step - loss: 0.0674 - acc: 0.9801
```

5. Evaluate the model on the test set (5 points)

Do NOT use the test set until now. Make sure that your model parameters and hyper-parameters are independent of the test set.

In [16]:

```
# Evaluate your model performance (testing accuracy) on testing data.
loss__acc = model.evaluate(x_test, y_test_vec)
print('loss = ' + str(loss__acc[0]))
print('accuracy = ' + str(loss__acc[1]))
```

```
313/313 [==============================] - 6s 17ms/step - loss: 4.0895 - acc: 0.6462
loss = 4.089514255523682
accuracy = 0.6462000012397766
```

In [ ]:

```

```

CNN model without Batch Normalization and without dropout layer Accuracy: 64%

6. Building model with new structure (25 points)

- In this section, you can build your model with adding new layers (e.g, BN layer or dropout layer, ...).
- If you want to regularize a `Conv/Dense layer`, you should place a `Dropout layer` before the `Conv/Dense layer`.
- You can try to compare their loss curve and testing accuracy and analyze your findings.
- You need to try at lease two different model structures.

Model 2: Added both Batch Normalization and DropOut layers

In [17]:

```
# Added both Batch-Normalization and DropoutLayers
model1=Sequential()
model1.add(Conv2D(32,(3,3),padding='valid',input_shape=(32,32,3)))
model1.add(BatchNormalization())
model1.add(Activation('relu'))
model1.add(MaxPooling2D((2,2)))

model1.add(Conv2D(64,(4,4),padding='valid'))
model1.add(BatchNormalization())
model1.add(Activation('relu'))
model1.add(MaxPooling2D((2,2)))

model1.add(Flatten())
model1.add(Dropout(0.5))
model1.add(Dense(256))
model1.add(BatchNormalization())
model1.add(Activation("relu"))
model1.add(Dropout(0.5))
model1.add(Dense(10))
model1.add(Activation('softmax'))

model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 30, 30, 32)        896

 batch_normalization (BatchN  (None, 30, 30, 32)       128
 ormalization)
```

```
 activation_4 (Activation)   (None, 30, 30, 32)        0

 max_pooling2d_2 (MaxPooling  (None, 15, 15, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 12, 12, 64)        32832

 batch_normalization_1 (Batc  (None, 12, 12, 64)       256
 hNormalization)

 activation_5 (Activation)   (None, 12, 12, 64)        0

 max_pooling2d_3 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 flatten_1 (Flatten)         (None, 2304)              0

 dropout (Dropout)           (None, 2304)              0

 dense_2 (Dense)             (None, 256)               590080

 batch_normalization_2 (Batc  (None, 256)              1024
 hNormalization)

 activation_6 (Activation)   (None, 256)               0

 dropout_1 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 10)                2570

 activation_7 (Activation)   (None, 10)                0

=================================================================
Total params: 627,786
Trainable params: 627,082
Non-trainable params: 704
_________________________________________________________________
```

In [18]:

```
learning_rate = 1e-4

model1.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(learning_rate=learning_rate),
              metrics=['acc'])
```

In [19]:

```
history1 = model1.fit(x_tr,y_tr, epochs=20, validation_data=(x_val,y_val))
```

```
Epoch 1/20
1250/1250 [==============================] - 113s 90ms/step - loss: 2.0145 - acc: 0.3176 - val_loss: 1.3903 - v
al_acc: 0.5080
Epoch 2/20
1250/1250 [==============================] - 109s 87ms/step - loss: 1.6151 - acc: 0.4331 - val_loss: 1.2402 - v
al_acc: 0.5656
Epoch 3/20
1250/1250 [==============================] - 112s 89ms/step - loss: 1.4524 - acc: 0.4868 - val_loss: 1.3117 - v
al_acc: 0.5533
Epoch 4/20
1250/1250 [==============================] - 110s 88ms/step - loss: 1.3420 - acc: 0.5264 - val_loss: 1.2154 - v
al_acc: 0.5649
Epoch 5/20
1250/1250 [==============================] - 110s 88ms/step - loss: 1.2779 - acc: 0.5521 - val_loss: 1.1173 - v
al_acc: 0.6026
Epoch 6/20
1250/1250 [==============================] - 106s 85ms/step - loss: 1.2276 - acc: 0.5674 - val_loss: 1.1475 - v
al_acc: 0.5943
Epoch 7/20
1250/1250 [==============================] - 110s 88ms/step - loss: 1.1835 - acc: 0.5829 - val_loss: 1.0942 - v
al_acc: 0.6227
Epoch 8/20
1250/1250 [==============================] - 110s 88ms/step - loss: 1.1537 - acc: 0.5997 - val_loss: 0.9716 - v
al_acc: 0.6664
Epoch 9/20
1250/1250 [==============================] - 110s 88ms/step - loss: 1.1313 - acc: 0.6094 - val_loss: 0.9589 - v
al_acc: 0.6694
Epoch 10/20
1250/1250 [==============================] - 113s 91ms/step - loss: 1.1104 - acc: 0.6176 - val_loss: 0.9707 - v
al_acc: 0.6700
Epoch 11/20
1250/1250 [==============================] - 107s 86ms/step - loss: 1.0987 - acc: 0.6211 - val_loss: 1.0033 - v
al_acc: 0.6522
Epoch 12/20
1250/1250 [==============================] - 112s 90ms/step - loss: 1.0712 - acc: 0.6323 - val_loss: 1.0616 - v
al_acc: 0.6266
Epoch 13/20
1250/1250 [==============================] - 107s 86ms/step - loss: 1.0639 - acc: 0.6370 - val_loss: 0.9777 - v
al_acc: 0.6632
Epoch 14/20
1250/1250 [==============================] - 108s 86ms/step - loss: 1.0450 - acc: 0.6420 - val_loss: 0.9584 - v
al_acc: 0.6661
```

```
Epoch 15/20
1250/1250 [==============================] - 108s 86ms/step - loss: 1.0393 - acc: 0.6488 - val_loss: 0.9184 - v
al_acc: 0.6909
Epoch 16/20
1250/1250 [==============================] - 114s 91ms/step - loss: 1.0317 - acc: 0.6503 - val_loss: 0.9794 - v
al_acc: 0.6706
Epoch 17/20
1250/1250 [==============================] - 107s 86ms/step - loss: 1.0196 - acc: 0.6555 - val_loss: 0.9121 - v
al_acc: 0.6907
Epoch 18/20
1250/1250 [==============================] - 108s 87ms/step - loss: 1.0059 - acc: 0.6609 - val_loss: 0.9603 - v
al_acc: 0.6694
Epoch 19/20
1250/1250 [==============================] - 109s 87ms/step - loss: 1.0020 - acc: 0.6600 - val_loss: 0.8836 - v
al_acc: 0.6945
Epoch 20/20
1250/1250 [==============================] - 108s 86ms/step - loss: 0.9966 - acc: 0.6617 - val_loss: 0.9216 - v
al_acc: 0.6796
```

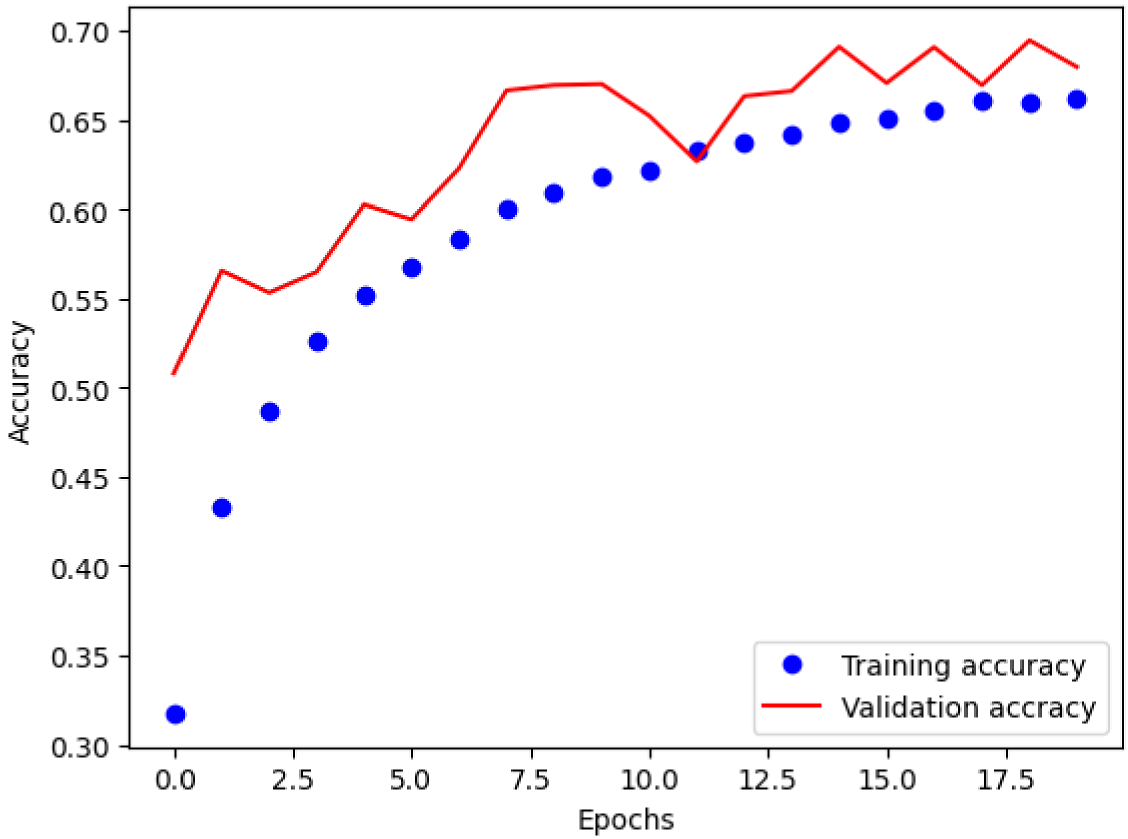
In [32]:

```
import matplotlib.pyplot as plt
%matplotlib inline

acc = history1.history['acc']
val_acc = history1.history['val_acc']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [33]:

```
val_acc = history1.history['val_acc']
train_acc=history1.history['acc']
avg_val_acc = sum(val_acc)/len(val_acc)
avg_train_acc=sum(train_acc)/len(train_acc)
print("Average validation accuracy:", avg_val_acc)
print("Average train accuracy:", avg_train_acc)
```

```
Average validation accuracy: 0.6360500007867813
Average train accuracy: 0.5881225034594536
```

In [35]:

```
#<Compile your model again (using the same hyper-parameters you tuned above)>
learning_rate = 1e-4

model1.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(learning_rate=learning_rate),
              metrics=['acc'])
```

In [36]:

```
history2 = model1.fit(x_train,y_train_vec, epochs=25)
```

```
Epoch 1/25
1563/1563 [==============================] - 134s 85ms/step - loss: 0.9181 - acc: 0.7050
Epoch 2/25
1563/1563 [==============================] - 131s 84ms/step - loss: 0.9160 - acc: 0.7065
Epoch 3/25
1563/1563 [==============================] - 129s 82ms/step - loss: 0.9140 - acc: 0.7076
Epoch 4/25
1563/1563 [==============================] - 131s 84ms/step - loss: 0.9066 - acc: 0.7052
Epoch 5/25
1563/1563 [==============================] - 130s 83ms/step - loss: 0.9126 - acc: 0.7078
Epoch 6/25
1563/1563 [==============================] - 134s 86ms/step - loss: 0.9090 - acc: 0.7104
Epoch 7/25
1563/1563 [==============================] - 131s 84ms/step - loss: 0.9077 - acc: 0.7081
Epoch 8/25
1563/1563 [==============================] - 132s 85ms/step - loss: 0.9054 - acc: 0.7094
Epoch 9/25
1563/1563 [==============================] - 132s 84ms/step - loss: 0.9065 - acc: 0.7097
Epoch 10/25
1563/1563 [==============================] - 130s 83ms/step - loss: 0.8966 - acc: 0.7119
Epoch 11/25
1563/1563 [==============================] - 127s 81ms/step - loss: 0.9068 - acc: 0.7102
Epoch 12/25
1563/1563 [==============================] - 129s 83ms/step - loss: 0.8982 - acc: 0.7130
Epoch 13/25
1563/1563 [==============================] - 129s 82ms/step - loss: 0.8958 - acc: 0.7134
Epoch 14/25
1563/1563 [==============================] - 130s 83ms/step - loss: 0.8951 - acc: 0.7159
Epoch 15/25
1563/1563 [==============================] - 129s 82ms/step - loss: 0.8968 - acc: 0.7114
Epoch 16/25
1563/1563 [==============================] - 130s 83ms/step - loss: 0.8969 - acc: 0.7163
Epoch 17/25
1563/1563 [==============================] - 130s 83ms/step - loss: 0.8873 - acc: 0.7178
Epoch 18/25
1563/1563 [==============================] - 128s 82ms/step - loss: 0.8872 - acc: 0.7160
Epoch 19/25
1563/1563 [==============================] - 130s 83ms/step - loss: 0.8934 - acc: 0.7164
Epoch 20/25
1563/1563 [==============================] - 128s 82ms/step - loss: 0.8872 - acc: 0.7144
Epoch 21/25
```

```
1563/1563 [==============================] - 131s 84ms/step - loss: 0.8872 - acc: 0.7177
Epoch 22/25
1563/1563 [==============================] - 131s 84ms/step - loss: 0.8798 - acc: 0.7172
Epoch 23/25
1563/1563 [==============================] - 130s 83ms/step - loss: 0.8924 - acc: 0.7177
Epoch 24/25
1563/1563 [==============================] - 128s 82ms/step - loss: 0.8811 - acc: 0.7178
Epoch 25/25
1563/1563 [==============================] - 130s 83ms/step - loss: 0.8818 - acc: 0.7196
```

In [37]:

```
loss__acc = model1.evaluate(x_test, y_test_vec)
print('loss = ' + str(loss__acc[0]))
print('accuracy = ' + str(loss__acc[1]))
```

```
313/313 [==============================] - 11s 36ms/step - loss: 0.7722 - acc: 0.7559
loss = 0.7721715569496155
accuracy = 0.7559000253677368
```

CNN model with Dropout Layers and Batch Normalization: Accuracy=75.5%**

Third Model with only Dropout Layers

In [38]:

```
# Only dropout Layers
model2=Sequential()
model2.add(Conv2D(32,(3,3),padding='valid',input_shape=(32,32,3)))
#model2.add(BatchNormalization())
model2.add(Activation('relu'))
model2.add(MaxPooling2D((2,2)))

model2.add(Conv2D(64,(4,4),padding='valid'))
#model2.add(BatchNormalization())
model2.add(Activation('relu'))
model2.add(MaxPooling2D((2,2)))

model2.add(Flatten())
model2.add(Dropout(0.5))
model2.add(Dense(256))
#model2.add(BatchNormalization())
model2.add(Activation("relu"))
model2.add(Dropout(0.5))
model2.add(Dense(10))
model2.add(Activation('softmax'))
```

```
model2.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_6 (Conv2D)           (None, 30, 30, 32)        896       
                                                                 
 activation_12 (Activation)  (None, 30, 30, 32)        0         
                                                                 
 max_pooling2d_6 (MaxPooling  (None, 15, 15, 32)       0         
 2D)                                                             
                                                                 
 conv2d_7 (Conv2D)           (None, 12, 12, 64)        32832     
                                                                 
 activation_13 (Activation)  (None, 12, 12, 64)        0         
                                                                 
 max_pooling2d_7 (MaxPooling  (None, 6, 6, 64)         0         
 2D)                                                             
                                                                 
 flatten_3 (Flatten)         (None, 2304)              0         
                                                                 
 dropout_4 (Dropout)         (None, 2304)              0         
                                                                 
 dense_6 (Dense)             (None, 256)               590080    
                                                                 
 activation_14 (Activation)  (None, 256)               0         
                                                                 
 dropout_5 (Dropout)         (None, 256)               0         
                                                                 
 dense_7 (Dense)             (None, 10)                2570      
                                                                 
 activation_15 (Activation)  (None, 10)                0         
                                                                 
=================================================================
Total params: 626,378
Trainable params: 626,378
Non-trainable params: 0
_________________________________________________________________
```

In [39]:

```
learning_rate = 1e-4

model2.compile(loss='categorical_crossentropy',
```

```
              optimizer=optimizers.RMSprop(learning_rate=learning_rate),
              metrics=['acc'])
```

In [40]:
```
history2 = model2.fit(x_tr,y_tr, epochs=20, validation_data=(x_val,y_val))
```

```
Epoch 1/20
1250/1250 [==============================] - 96s 76ms/step - loss: 3.6328 - acc: 0.1258 - val_loss: 2.2506 - va
l_acc: 0.1436
Epoch 2/20
1250/1250 [==============================] - 95s 76ms/step - loss: 2.2195 - acc: 0.1692 - val_loss: 2.1200 - va
l_acc: 0.2311
Epoch 3/20
1250/1250 [==============================] - 93s 74ms/step - loss: 2.1429 - acc: 0.2181 - val_loss: 2.1208 - va
l_acc: 0.2291
Epoch 4/20
1250/1250 [==============================] - 94s 75ms/step - loss: 2.0764 - acc: 0.2378 - val_loss: 1.9844 - va
l_acc: 0.2804
Epoch 5/20
1250/1250 [==============================] - 94s 75ms/step - loss: 2.0042 - acc: 0.2615 - val_loss: 1.8683 - va
l_acc: 0.3169
Epoch 6/20
1250/1250 [==============================] - 93s 74ms/step - loss: 1.9401 - acc: 0.2841 - val_loss: 1.8277 - va
l_acc: 0.3367
Epoch 7/20
1250/1250 [==============================] - 88s 70ms/step - loss: 1.8713 - acc: 0.3117 - val_loss: 1.7387 - va
l_acc: 0.3705
Epoch 8/20
1250/1250 [==============================] - 91s 72ms/step - loss: 1.8190 - acc: 0.3335 - val_loss: 1.6368 - va
l_acc: 0.3992
Epoch 9/20
1250/1250 [==============================] - 90s 72ms/step - loss: 1.7837 - acc: 0.3436 - val_loss: 1.6016 - va
l_acc: 0.4192
Epoch 10/20
1250/1250 [==============================] - 88s 70ms/step - loss: 1.7481 - acc: 0.3577 - val_loss: 1.5911 - va
l_acc: 0.4249
Epoch 11/20
1250/1250 [==============================] - 90s 72ms/step - loss: 1.7293 - acc: 0.3657 - val_loss: 1.5698 - va
l_acc: 0.4215
Epoch 12/20
1250/1250 [==============================] - 90s 72ms/step - loss: 1.7020 - acc: 0.3787 - val_loss: 1.5579 - va
l_acc: 0.4261
Epoch 13/20
1250/1250 [==============================] - 88s 71ms/step - loss: 1.6930 - acc: 0.3855 - val_loss: 1.6115 - va
l_acc: 0.3980
```

```
Epoch 14/20
1250/1250 [==============================] - 91s 72ms/step - loss: 1.6730 - acc: 0.3915 - val_loss: 1.5527 - va
l_acc: 0.4386
Epoch 15/20
1250/1250 [==============================] - 95s 76ms/step - loss: 1.6659 - acc: 0.3911 - val_loss: 1.5553 - va
l_acc: 0.4230
Epoch 16/20
1250/1250 [==============================] - 94s 75ms/step - loss: 1.6521 - acc: 0.3992 - val_loss: 1.6046 - va
l_acc: 0.4290
Epoch 17/20
1250/1250 [==============================] - 90s 72ms/step - loss: 1.6369 - acc: 0.4051 - val_loss: 1.4799 - va
l_acc: 0.4706
Epoch 18/20
1250/1250 [==============================] - 90s 72ms/step - loss: 1.6361 - acc: 0.4101 - val_loss: 1.4953 - va
l_acc: 0.4716
Epoch 19/20
1250/1250 [==============================] - 94s 75ms/step - loss: 1.6139 - acc: 0.4146 - val_loss: 1.4807 - va
l_acc: 0.4545
Epoch 20/20
1250/1250 [==============================] - 91s 72ms/step - loss: 1.6090 - acc: 0.4266 - val_loss: 1.4758 - va
l_acc: 0.4580
```

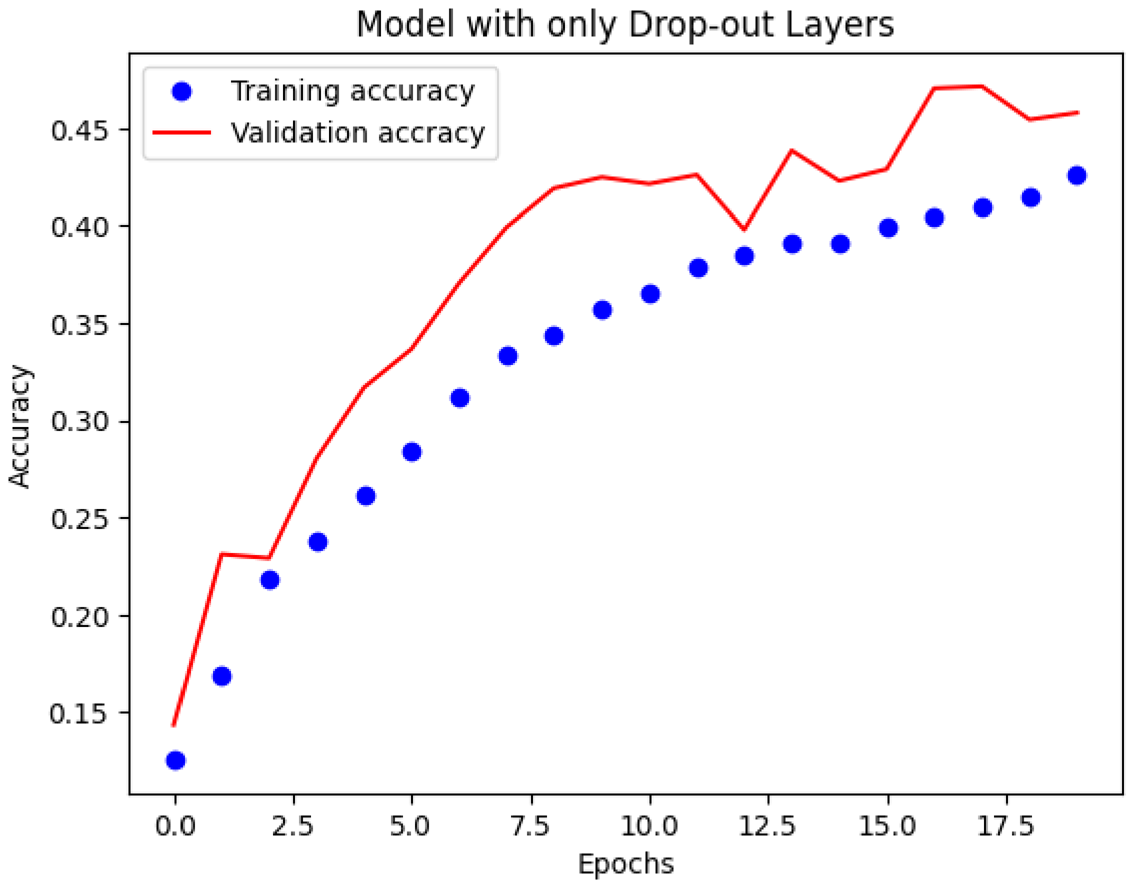
In [41]:

```
import matplotlib.pyplot as plt
%matplotlib inline

acc = history2.history['acc']
val_acc = history2.history['val_acc']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title(" Model with only Drop-out Layers ")
plt.legend()
plt.show()
```



In [42]:

```
val_acc = history2.history['val_acc']
train_acc=history2.history['acc']
avg_val_acc = sum(val_acc)/len(val_acc)
avg_train_acc=sum(train_acc)/len(train_acc)
print("Average validation accuracy:", avg_val_acc)
print("Average train accuracy:", avg_train_acc)
```

```
Average validation accuracy: 0.3771249994635582
Average train accuracy: 0.3305499993264675
```

In [43]:

```
#<Compile your model again (using the same hyper-parameters you tuned above)>
learning_rate = 1e-4

model2.compile(loss='categorical_crossentropy',
```

```
              optimizer=optimizers.RMSprop(learning_rate=learning_rate),
              metrics=['acc'])
```

In [44]:

```
history2 = model2.fit(x_train,y_train_vec, epochs=25)
```

```
Epoch 1/25
1563/1563 [==============================] - 108s 69ms/step - loss: 1.5971 - acc: 0.4299
Epoch 2/25
1563/1563 [==============================] - 106s 68ms/step - loss: 1.5878 - acc: 0.4369
Epoch 3/25
1563/1563 [==============================] - 104s 67ms/step - loss: 1.5621 - acc: 0.4452
Epoch 4/25
1563/1563 [==============================] - 105s 67ms/step - loss: 1.5595 - acc: 0.4526
Epoch 5/25
1563/1563 [==============================] - 105s 67ms/step - loss: 1.5488 - acc: 0.4576
Epoch 6/25
1563/1563 [==============================] - 106s 68ms/step - loss: 1.5337 - acc: 0.4638
Epoch 7/25
1563/1563 [==============================] - 106s 68ms/step - loss: 1.5201 - acc: 0.4690
Epoch 8/25
1563/1563 [==============================] - 105s 67ms/step - loss: 1.5165 - acc: 0.4700
Epoch 9/25
1563/1563 [==============================] - 106s 68ms/step - loss: 1.5080 - acc: 0.4760
Epoch 10/25
1563/1563 [==============================] - 107s 68ms/step - loss: 1.5072 - acc: 0.4776
Epoch 11/25
1563/1563 [==============================] - 108s 69ms/step - loss: 1.5048 - acc: 0.4789
Epoch 12/25
1563/1563 [==============================] - 106s 68ms/step - loss: 1.4991 - acc: 0.4831
Epoch 13/25
1563/1563 [==============================] - 105s 67ms/step - loss: 1.4991 - acc: 0.4866
Epoch 14/25
1563/1563 [==============================] - 107s 68ms/step - loss: 1.4954 - acc: 0.4861
Epoch 15/25
1563/1563 [==============================] - 108s 69ms/step - loss: 1.5015 - acc: 0.4841
Epoch 16/25
1563/1563 [==============================] - 107s 68ms/step - loss: 1.5017 - acc: 0.4865
Epoch 17/25
1563/1563 [==============================] - 107s 69ms/step - loss: 1.5070 - acc: 0.4842
Epoch 18/25
1563/1563 [==============================] - 106s 68ms/step - loss: 1.5064 - acc: 0.4856
Epoch 19/25
1563/1563 [==============================] - 107s 69ms/step - loss: 1.5170 - acc: 0.4865
Epoch 20/25
```

```
1563/1563 [==============================] - 108s 69ms/step - loss: 1.5045 - acc: 0.4868
Epoch 21/25
1563/1563 [==============================] - 108s 69ms/step - loss: 1.5099 - acc: 0.4872
Epoch 22/25
1563/1563 [==============================] - 107s 69ms/step - loss: 1.5057 - acc: 0.4926
Epoch 23/25
1563/1563 [==============================] - 105s 67ms/step - loss: 1.4991 - acc: 0.4936
Epoch 24/25
1563/1563 [==============================] - 108s 69ms/step - loss: 1.4993 - acc: 0.4917
Epoch 25/25
1563/1563 [==============================] - 107s 68ms/step - loss: 1.5038 - acc: 0.4925
```

In [45]:

```
loss__acc = model2.evaluate(x_test, y_test_vec)
print('loss = ' + str(loss__acc[0]))
print('accuracy = ' + str(loss__acc[1]))
```

```
313/313 [==============================] - 6s 18ms/step - loss: 1.3696 - acc: 0.5295
loss = 1.3696386814117432
accuracy = 0.5295000076293945
```

CNN model with only Dropout Layers: Accuracy= 52.9%

Conclusion:

The testing accuracy using the model with complete data

- **Basic CNN model(No dropout layers, No BatchNormalization): 0.64**
- **CNN Model with droput layers and batchNormalization is : 0.755**
- **CNN model with only dropout layers: 0.529**

The accuracy of a model tested on complete data varies with the type of architecture used. In this study, we evaluated the performance of three CNN models with varying architectures: a basic CNN model without dropout layers or BatchNormalization achieved an accuracy of 0.64, while a CNN model with dropout layers and BatchNormalization achieved an accuracy of 0.755. In contrast, a CNN model with only dropout layers achieved an accuracy of 0.529, indicating poor performance compared to the other two models.

It is evident from our results that the inclusion of BatchNormalization and dropout layers had a positive impact on the accuracy of the CNN model. Specifically, **the model that included BatchNormalization** had the highest accuracy, thereby demonstrating the effectiveness of BatchNormalization in enhancing model performance.

In [ ]:

```

```