Cifar_10_Final_Project

December 13, 2019

1 Final Project - Cifar 10

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. We will be building a model that predicts/classifies what classes the images fall into. For this project, we will be using Keras.

1.1 Getting Data

To begin, we will load the cifar-10 dataset. X_train and y_train will contain information for 50,000 training images. X_test and y_test will contain information for 10,000 test images. We will leave the test images alone and only test on the model that produces the best result on the validation set. We also create an array that labels what the 10 classes should be.

Additionally, we will create a validation set once we train the model, randomly assigning 20% of the data to be validation data.

[0]:
```
from tensorflow import keras
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from keras.utils import print_summary, to_categorical
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from keras import regularizers

import numpy as np
import os

import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix


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

label_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog',␣
↪'horse', 'ship', 'truck']
```

1.2 Useful Helper Functions

Below, we will create a few helper functions that will help us evaluate our model.

1.2.1 Validation Helper

This helper function will help plot the training and validation loss graph to see if we're underfitting or overfitting the data. We will also plot the training and validation accuracy data to compare with the loss.

1.2.2 Test Set Evaluation

The first function (get_class_from_softmax) will help us turn an array of probabilities of likelihood of each class (softmax_list) to a the index of the maximum probability, which we can convert to a class using the label_names array (convert_to_labels function).

We will then use the helpers above in the get_metrics function, which will produce a classification report and confusion matrix for the test set.

[0]:
```
# Training vs validation loss graph

import matplotlib.pyplot as plt

def plot_graphs(history):
  # Plot training & validation loss values
  plt.plot(history.history['loss'])
  plt.plot(history.history['val_loss'])
  plt.title('Model loss')
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Validation'], loc='upper left')
  plt.show()

  # Plot training & validation accuracy values
  plt.plot(history.history['acc'])
  plt.plot(history.history['val_acc'])
  plt.title('Model accuracy')
  plt.ylabel('Accuracy')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Validation'], loc='upper left')
  plt.show()
```

[0]:
```
## Here we'll use the model to classify the test set!
import pandas as pd

def get_class_from_softmax(softmax_list):
    maximum = 0
    max_index = 0
```

```
    for i in range(len(softmax_list)):
      if softmax_list[i] > maximum:
        maximum = softmax_list[i]
        max_index = i

    return max_index

def convert_to_labels(A):
  labels = list()
  for row in A:
    labels.append(label_names[get_class_from_softmax(row)])
  return labels

def get_metrics(model_name):
  model = load_model(model_name)

  print(model.evaluate(x=x_test, y=to_categorical(y_test, num_classes)))

  predicted = model.predict(x_test)


  predicted_labels = convert_to_labels(predicted)
  actual_labels = convert_to_labels(to_categorical(y_test, num_classes))

  # Generate classification report
  print(classification_report(y_true=actual_labels,
                              y_pred=predicted_labels,
                              labels=label_names))

  # Generate the confusion matrix
  matrix = pd.DataFrame(
      confusion_matrix(actual_labels, predicted_labels, labels=label_names),
      index=label_names,
      columns=['predicted - airplane',
               'predicted - automobile',
               'predicted - bird',
               'predicted - cat',
               'predicted - deer',
               'predicted - dog',
               'predicted - frog',
               'predicted - horse',
               'predicted - ship',
               'predicted - truck']
  )
  pd.set_option('display.max_columns', None)

  print(matrix)
```

1.3 Model Creating, Training, and Validating

We will begin with a batch size of 32 and 25 epochs.

Additionally, note that when we're training(fitting) the model, we add the validation split of 0.2, which splits the training data randomly into 80% training and 20% validation so we can see how well the model is performing. We will be using this split for all models below.

```
history = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)
```

[0]:
```
batch_size = 32
num_classes = 10
epochs = 25

shape = (32, 32, 3)
```

1.4 Model #1

For our first model, we will begin by creating a 6 layer CNN. For the optimizer, we will arbitrarily choose "RMSprop". We will test different optimizers as we proceed as well as additional layers.

[7]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 input_shape=shape))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3),
```

```
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compile the model
opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

print(model.summary())

#Train model
history = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_1.h5', overwrite=True)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow_core/python/keras/initializers.py:119: calling
RandomUniform.__init__ (from tensorflow.python.ops.init_ops) with dtype is
deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling
BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops)
with constraint is deprecated and will be removed in a future version.
Instructions for updating:
```

```
If using Keras pass *_constraint arguments to layers.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
activation (Activation)      (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
activation_4 (Activation)    (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
activation_5 (Activation)    (None, 8, 8, 128)         0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 256)               524544
_________________________________________________________________
activation_6 (Activation)    (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                2570
_________________________________________________________________
activation_7 (Activation)    (None, 10)                0
=================================================================
Total params: 814,122
Trainable params: 814,122
Non-trainable params: 0
```

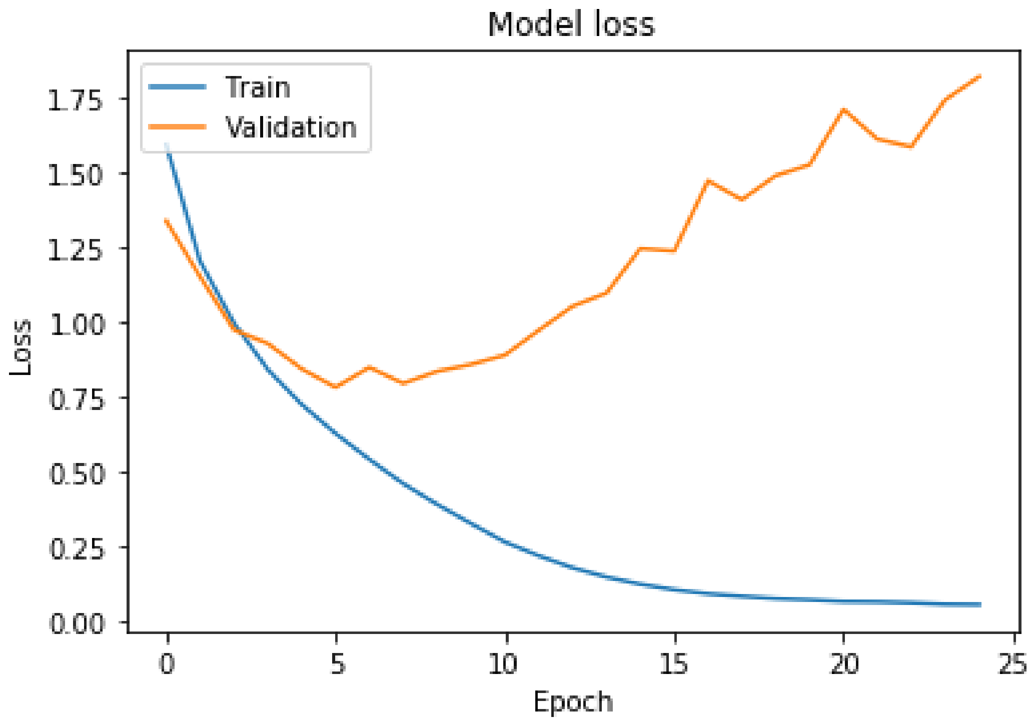
```
_________________________________________________________________
None
Train on 40000 samples, validate on 10000 samples
Epoch 1/25
40000/40000 - 11s - loss: 1.5936 - acc: 0.4250 - val_loss: 1.3381 - val_acc:
0.5228
Epoch 2/25
40000/40000 - 7s - loss: 1.2037 - acc: 0.5775 - val_loss: 1.1519 - val_acc:
0.6031
Epoch 3/25
40000/40000 - 7s - loss: 0.9967 - acc: 0.6511 - val_loss: 0.9737 - val_acc:
0.6571
Epoch 4/25
40000/40000 - 7s - loss: 0.8404 - acc: 0.7083 - val_loss: 0.9273 - val_acc:
0.6816
Epoch 5/25
40000/40000 - 7s - loss: 0.7237 - acc: 0.7505 - val_loss: 0.8428 - val_acc:
0.7139
Epoch 6/25
40000/40000 - 7s - loss: 0.6264 - acc: 0.7854 - val_loss: 0.7811 - val_acc:
0.7359
Epoch 7/25
40000/40000 - 7s - loss: 0.5395 - acc: 0.8138 - val_loss: 0.8480 - val_acc:
0.7216
Epoch 8/25
40000/40000 - 7s - loss: 0.4589 - acc: 0.8409 - val_loss: 0.7943 - val_acc:
0.7477
Epoch 9/25
40000/40000 - 7s - loss: 0.3891 - acc: 0.8666 - val_loss: 0.8347 - val_acc:
0.7446
Epoch 10/25
40000/40000 - 7s - loss: 0.3249 - acc: 0.8871 - val_loss: 0.8575 - val_acc:
0.7444
Epoch 11/25
40000/40000 - 7s - loss: 0.2630 - acc: 0.9092 - val_loss: 0.8884 - val_acc:
0.7497
Epoch 12/25
40000/40000 - 7s - loss: 0.2169 - acc: 0.9259 - val_loss: 0.9716 - val_acc:
0.7481
Epoch 13/25
40000/40000 - 7s - loss: 0.1766 - acc: 0.9393 - val_loss: 1.0522 - val_acc:
0.7475
Epoch 14/25
40000/40000 - 7s - loss: 0.1458 - acc: 0.9492 - val_loss: 1.0967 - val_acc:
0.7553
Epoch 15/25
40000/40000 - 7s - loss: 0.1221 - acc: 0.9569 - val_loss: 1.2455 - val_acc:
0.7461
```

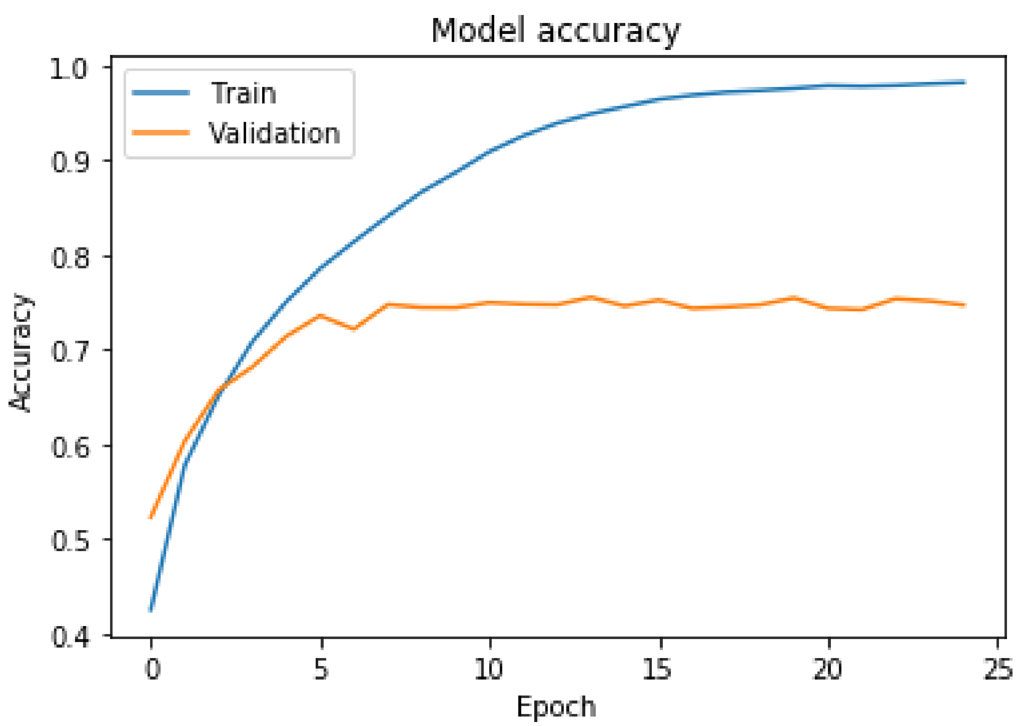
```
Epoch 16/25
40000/40000 - 7s - loss: 0.1035 - acc: 0.9645 - val_loss: 1.2370 - val_acc:
0.7523
Epoch 17/25
40000/40000 - 7s - loss: 0.0901 - acc: 0.9689 - val_loss: 1.4726 - val_acc:
0.7436
Epoch 18/25
40000/40000 - 7s - loss: 0.0814 - acc: 0.9721 - val_loss: 1.4091 - val_acc:
0.7451
Epoch 19/25
40000/40000 - 7s - loss: 0.0739 - acc: 0.9738 - val_loss: 1.4897 - val_acc:
0.7474
Epoch 20/25
40000/40000 - 7s - loss: 0.0698 - acc: 0.9760 - val_loss: 1.5266 - val_acc:
0.7546
Epoch 21/25
40000/40000 - 7s - loss: 0.0641 - acc: 0.9789 - val_loss: 1.7113 - val_acc:
0.7436
Epoch 22/25
40000/40000 - 7s - loss: 0.0631 - acc: 0.9782 - val_loss: 1.6118 - val_acc:
0.7422
Epoch 23/25
40000/40000 - 7s - loss: 0.0599 - acc: 0.9791 - val_loss: 1.5860 - val_acc:
0.7538
Epoch 24/25
40000/40000 - 7s - loss: 0.0548 - acc: 0.9809 - val_loss: 1.7428 - val_acc:
0.7514
Epoch 25/25
40000/40000 - 7s - loss: 0.0535 - acc: 0.9822 - val_loss: 1.8211 - val_acc:
0.7472
```

1.4.1 Validating Model #1

We can see from the plot below that we are heavily overfitting the data. To prevent this, we will try adding regularization.

[10]:
```
plot_graphs(history)
```


Model loss
Train
Validation
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Loss
0
5
10
15
20
25
Epoch


Model accuracy
Train
Validation
1.0
0.9
0.8
0.7
0.6
0.5
0.4
Accuracy
0
5
10
15
20
25
Epoch

1.5 Model #2 - Regularization with Weight Decay

1.5.1 We will try adding a weight decay of 0.0005 (default)

[0]:
```
from keras import regularizers

model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 input_shape=shape,
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dense(num_classes))
```

```python
model.add(Activation('softmax'))

# Compile the model
opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history1 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_2_part1.h5', overwrite=True)
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/25
40000/40000 - 20s - loss: 2.8412 - acc: 0.4051 - val_loss: 2.2614 - val_acc:
0.5002
Epoch 2/25
40000/40000 - 12s - loss: 2.0279 - acc: 0.5384 - val_loss: 1.8188 - val_acc:
0.5744
Epoch 3/25
40000/40000 - 13s - loss: 1.6899 - acc: 0.6036 - val_loss: 1.5477 - val_acc:
0.6357
Epoch 4/25
40000/40000 - 12s - loss: 1.4689 - acc: 0.6517 - val_loss: 1.4261 - val_acc:
0.6608
Epoch 5/25
40000/40000 - 13s - loss: 1.3099 - acc: 0.6872 - val_loss: 1.3336 - val_acc:
0.6715
Epoch 6/25
40000/40000 - 13s - loss: 1.1912 - acc: 0.7153 - val_loss: 1.2332 - val_acc:
0.7037
Epoch 7/25
40000/40000 - 13s - loss: 1.0914 - acc: 0.7416 - val_loss: 1.1738 - val_acc:
0.7123
Epoch 8/25
40000/40000 - 13s - loss: 1.0148 - acc: 0.7612 - val_loss: 1.1587 - val_acc:
0.7119
Epoch 9/25
40000/40000 - 13s - loss: 0.9502 - acc: 0.7790 - val_loss: 1.1558 - val_acc:
0.7120
```

```
Epoch 10/25
40000/40000 - 13s - loss: 0.8894 - acc: 0.7993 - val_loss: 1.0117 - val_acc:
0.7581
Epoch 11/25
40000/40000 - 12s - loss: 0.8411 - acc: 0.8103 - val_loss: 1.0348 - val_acc:
0.7503
Epoch 12/25
40000/40000 - 12s - loss: 0.7925 - acc: 0.8259 - val_loss: 1.0098 - val_acc:
0.7585
Epoch 13/25
40000/40000 - 13s - loss: 0.7529 - acc: 0.8389 - val_loss: 0.9925 - val_acc:
0.7647
Epoch 14/25
40000/40000 - 13s - loss: 0.7141 - acc: 0.8512 - val_loss: 0.9833 - val_acc:
0.7700
Epoch 15/25
40000/40000 - 13s - loss: 0.6782 - acc: 0.8624 - val_loss: 1.0224 - val_acc:
0.7660
Epoch 16/25
40000/40000 - 12s - loss: 0.6439 - acc: 0.8727 - val_loss: 1.0515 - val_acc:
0.7622
Epoch 17/25
40000/40000 - 12s - loss: 0.6134 - acc: 0.8838 - val_loss: 1.0397 - val_acc:
0.7640
Epoch 18/25
40000/40000 - 13s - loss: 0.5812 - acc: 0.8935 - val_loss: 1.0686 - val_acc:
0.7664
Epoch 19/25
40000/40000 - 13s - loss: 0.5579 - acc: 0.9033 - val_loss: 1.1098 - val_acc:
0.7611
Epoch 20/25
40000/40000 - 13s - loss: 0.5293 - acc: 0.9123 - val_loss: 1.1008 - val_acc:
0.7646
Epoch 21/25
40000/40000 - 12s - loss: 0.5046 - acc: 0.9221 - val_loss: 1.0843 - val_acc:
0.7717
Epoch 22/25
40000/40000 - 13s - loss: 0.4880 - acc: 0.9258 - val_loss: 1.1731 - val_acc:
0.7588
Epoch 23/25
40000/40000 - 12s - loss: 0.4678 - acc: 0.9329 - val_loss: 1.2208 - val_acc:
0.7593
Epoch 24/25
40000/40000 - 12s - loss: 0.4534 - acc: 0.9373 - val_loss: 1.2810 - val_acc:
0.7424
Epoch 25/25
40000/40000 - 13s - loss: 0.4402 - acc: 0.9418 - val_loss: 1.1663 - val_acc:
0.7663
```

Now, we can see that the training and validation loss are closer to each other! We are overfitting a bit less; however, we can improve this model. Let's continue to try with different weight decays.

Weight decay of 0.00005

[0]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 input_shape=shape,
                 kernel_regularizer=regularizers.l2(0.00005)))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(0.00005)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(0.00005)))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(0.00005)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(0.00005)))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(0.00005)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dense(num_classes))
```

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

# Compile the model
opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history2 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_2_part2.h5', overwrite=True)
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/25
40000/40000 - 20s - loss: 1.6239 - acc: 0.4171 - val_loss: 1.3006 - val_acc:
0.5461
Epoch 2/25
40000/40000 - 13s - loss: 1.2038 - acc: 0.5789 - val_loss: 1.1822 - val_acc:
0.6013
Epoch 3/25
40000/40000 - 13s - loss: 0.9981 - acc: 0.6535 - val_loss: 0.9808 - val_acc:
0.6609
Epoch 4/25
40000/40000 - 13s - loss: 0.8509 - acc: 0.7084 - val_loss: 0.8734 - val_acc:
0.7075
Epoch 5/25
40000/40000 - 13s - loss: 0.7345 - acc: 0.7496 - val_loss: 0.9499 - val_acc:
0.6783
Epoch 6/25
40000/40000 - 13s - loss: 0.6365 - acc: 0.7861 - val_loss: 0.8492 - val_acc:
0.7271
Epoch 7/25
40000/40000 - 13s - loss: 0.5523 - acc: 0.8152 - val_loss: 0.7895 - val_acc:
0.7444
Epoch 8/25
40000/40000 - 12s - loss: 0.4699 - acc: 0.8444 - val_loss: 0.8096 - val_acc:
0.7415
Epoch 9/25
40000/40000 - 12s - loss: 0.4002 - acc: 0.8667 - val_loss: 0.8677 - val_acc:
0.7435
```

```
Epoch 10/25
40000/40000 - 13s - loss: 0.3332 - acc: 0.8898 - val_loss: 0.8790 - val_acc:
0.7421
Epoch 11/25
40000/40000 - 12s - loss: 0.2740 - acc: 0.9112 - val_loss: 0.9327 - val_acc:
0.7487
Epoch 12/25
40000/40000 - 13s - loss: 0.2242 - acc: 0.9281 - val_loss: 1.0310 - val_acc:
0.7477
Epoch 13/25
40000/40000 - 13s - loss: 0.1843 - acc: 0.9405 - val_loss: 1.1074 - val_acc:
0.7431
Epoch 14/25
40000/40000 - 13s - loss: 0.1541 - acc: 0.9504 - val_loss: 1.2450 - val_acc:
0.7426
Epoch 15/25
40000/40000 - 12s - loss: 0.1338 - acc: 0.9588 - val_loss: 1.3136 - val_acc:
0.7441
Epoch 16/25
40000/40000 - 12s - loss: 0.1156 - acc: 0.9651 - val_loss: 1.4019 - val_acc:
0.7397
Epoch 17/25
40000/40000 - 13s - loss: 0.1079 - acc: 0.9685 - val_loss: 1.4684 - val_acc:
0.7474
Epoch 18/25
40000/40000 - 13s - loss: 0.0973 - acc: 0.9714 - val_loss: 1.4947 - val_acc:
0.7443
Epoch 19/25
40000/40000 - 13s - loss: 0.0894 - acc: 0.9748 - val_loss: 1.6713 - val_acc:
0.7397
Epoch 20/25
40000/40000 - 13s - loss: 0.0859 - acc: 0.9751 - val_loss: 1.6257 - val_acc:
0.7533
Epoch 21/25
40000/40000 - 12s - loss: 0.0828 - acc: 0.9778 - val_loss: 1.6815 - val_acc:
0.7401
Epoch 22/25
40000/40000 - 13s - loss: 0.0773 - acc: 0.9786 - val_loss: 1.9042 - val_acc:
0.7441
Epoch 23/25
40000/40000 - 13s - loss: 0.0749 - acc: 0.9802 - val_loss: 1.9526 - val_acc:
0.7429
Epoch 24/25
40000/40000 - 13s - loss: 0.0721 - acc: 0.9812 - val_loss: 1.9188 - val_acc:
0.7399
Epoch 25/25
40000/40000 - 13s - loss: 0.0729 - acc: 0.9810 - val_loss: 1.8911 - val_acc:
0.7455
```

Weight decay of 0.05

[0]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 input_shape=shape,
                 kernel_regularizer=regularizers.l2(0.05)))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(0.05)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(0.05)))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(0.05)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(0.05)))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(0.05)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compile the model
opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=1e-6)
```

```python
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history3 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_2_part3.h5', overwrite=True)
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/25
40000/40000 - 20s - loss: 4.7571 - acc: 0.3735 - val_loss: 2.5774 - val_acc:
0.4506
Epoch 2/25
40000/40000 - 13s - loss: 2.2108 - acc: 0.4658 - val_loss: 1.9172 - val_acc:
0.5055
Epoch 3/25
40000/40000 - 13s - loss: 1.8150 - acc: 0.5124 - val_loss: 1.6739 - val_acc:
0.5514
Epoch 4/25
40000/40000 - 13s - loss: 1.6372 - acc: 0.5481 - val_loss: 1.6642 - val_acc:
0.5279
Epoch 5/25
40000/40000 - 13s - loss: 1.5275 - acc: 0.5727 - val_loss: 1.6020 - val_acc:
0.5425
Epoch 6/25
40000/40000 - 13s - loss: 1.4457 - acc: 0.5938 - val_loss: 1.4015 - val_acc:
0.6060
Epoch 7/25
40000/40000 - 13s - loss: 1.3846 - acc: 0.6139 - val_loss: 1.4191 - val_acc:
0.6046
Epoch 8/25
40000/40000 - 13s - loss: 1.3289 - acc: 0.6331 - val_loss: 1.3470 - val_acc:
0.6224
Epoch 9/25
40000/40000 - 13s - loss: 1.2845 - acc: 0.6452 - val_loss: 1.3618 - val_acc:
0.6209
Epoch 10/25
40000/40000 - 13s - loss: 1.2481 - acc: 0.6583 - val_loss: 1.2855 - val_acc:
0.6403
Epoch 11/25
```

```
40000/40000 - 13s - loss: 1.2156 - acc: 0.6690 - val_loss: 1.2581 - val_acc:
0.6544
Epoch 12/25
40000/40000 - 13s - loss: 1.1878 - acc: 0.6782 - val_loss: 1.1940 - val_acc:
0.6797
Epoch 13/25
40000/40000 - 13s - loss: 1.1588 - acc: 0.6870 - val_loss: 1.2840 - val_acc:
0.6388
Epoch 14/25
40000/40000 - 13s - loss: 1.1346 - acc: 0.6952 - val_loss: 1.1559 - val_acc:
0.6885
Epoch 15/25
40000/40000 - 13s - loss: 1.1138 - acc: 0.7029 - val_loss: 1.1870 - val_acc:
0.6811
Epoch 16/25
40000/40000 - 13s - loss: 1.0946 - acc: 0.7104 - val_loss: 1.1661 - val_acc:
0.6839
Epoch 17/25
40000/40000 - 13s - loss: 1.0748 - acc: 0.7166 - val_loss: 1.1481 - val_acc:
0.6918
Epoch 18/25
40000/40000 - 13s - loss: 1.0567 - acc: 0.7224 - val_loss: 1.1470 - val_acc:
0.6939
Epoch 19/25
40000/40000 - 13s - loss: 1.0417 - acc: 0.7302 - val_loss: 1.1716 - val_acc:
0.6861
Epoch 20/25
40000/40000 - 13s - loss: 1.0221 - acc: 0.7346 - val_loss: 1.1957 - val_acc:
0.6837
Epoch 21/25
40000/40000 - 13s - loss: 1.0045 - acc: 0.7424 - val_loss: 1.1217 - val_acc:
0.7051
Epoch 22/25
40000/40000 - 13s - loss: 0.9901 - acc: 0.7479 - val_loss: 1.1151 - val_acc:
0.7088
Epoch 23/25
40000/40000 - 13s - loss: 0.9783 - acc: 0.7533 - val_loss: 1.1005 - val_acc:
0.7129
Epoch 24/25
40000/40000 - 13s - loss: 0.9622 - acc: 0.7575 - val_loss: 1.1166 - val_acc:
0.7120
Epoch 25/25
40000/40000 - 13s - loss: 0.9474 - acc: 0.7643 - val_loss: 1.1319 - val_acc:
0.7060
```

Weight decay of 0

[0]:

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 input_shape=shape,
                 kernel_regularizer=regularizers.l2(0)))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(0)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(0)))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(0)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(0)))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(0)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compile the model
opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',
```

```
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history4 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_2_part4.h5', overwrite=True)
```

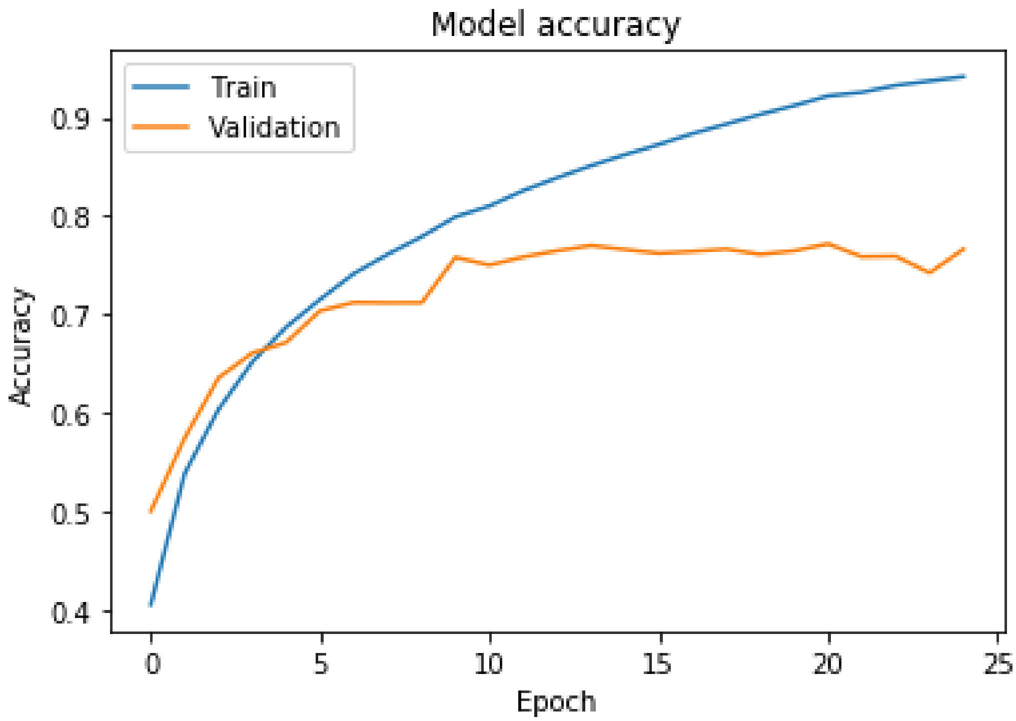
```
Train on 40000 samples, validate on 10000 samples
Epoch 1/25
40000/40000 - 20s - loss: 1.5688 - acc: 0.4353 - val_loss: 1.3902 - val_acc:
0.5000
Epoch 2/25
40000/40000 - 12s - loss: 1.1755 - acc: 0.5842 - val_loss: 1.1016 - val_acc:
0.6126
Epoch 3/25
40000/40000 - 12s - loss: 0.9641 - acc: 0.6640 - val_loss: 0.9293 - val_acc:
0.6826
Epoch 4/25
40000/40000 - 12s - loss: 0.8164 - acc: 0.7182 - val_loss: 0.8757 - val_acc:
0.7034
Epoch 5/25
40000/40000 - 12s - loss: 0.7024 - acc: 0.7575 - val_loss: 0.8251 - val_acc:
0.7245
Epoch 6/25
40000/40000 - 12s - loss: 0.6080 - acc: 0.7913 - val_loss: 0.7696 - val_acc:
0.7414
Epoch 7/25
40000/40000 - 12s - loss: 0.5219 - acc: 0.8204 - val_loss: 0.7703 - val_acc:
0.7437
Epoch 8/25
40000/40000 - 12s - loss: 0.4447 - acc: 0.8470 - val_loss: 0.8055 - val_acc:
0.7478
Epoch 9/25
40000/40000 - 12s - loss: 0.3731 - acc: 0.8709 - val_loss: 0.9227 - val_acc:
0.7297
Epoch 10/25
40000/40000 - 12s - loss: 0.3073 - acc: 0.8944 - val_loss: 0.8496 - val_acc:
0.7526
Epoch 11/25
40000/40000 - 12s - loss: 0.2524 - acc: 0.9119 - val_loss: 0.8732 - val_acc:
```

```
0.7534
Epoch 12/25
40000/40000 - 12s - loss: 0.2036 - acc: 0.9294 - val_loss: 0.9980 - val_acc:
0.7461
Epoch 13/25
40000/40000 - 12s - loss: 0.1658 - acc: 0.9430 - val_loss: 1.0733 - val_acc:
0.7465
Epoch 14/25
40000/40000 - 12s - loss: 0.1387 - acc: 0.9516 - val_loss: 1.0687 - val_acc:
0.7581
Epoch 15/25
40000/40000 - 12s - loss: 0.1155 - acc: 0.9596 - val_loss: 1.3045 - val_acc:
0.7529
Epoch 16/25
40000/40000 - 12s - loss: 0.1000 - acc: 0.9648 - val_loss: 1.2257 - val_acc:
0.7524
Epoch 17/25
40000/40000 - 12s - loss: 0.0868 - acc: 0.9699 - val_loss: 1.4208 - val_acc:
0.7408
Epoch 18/25
40000/40000 - 12s - loss: 0.0778 - acc: 0.9721 - val_loss: 1.4997 - val_acc:
0.7478
Epoch 19/25
40000/40000 - 12s - loss: 0.0729 - acc: 0.9749 - val_loss: 1.5484 - val_acc:
0.7434
Epoch 20/25
40000/40000 - 12s - loss: 0.0673 - acc: 0.9764 - val_loss: 1.6378 - val_acc:
0.7366
Epoch 21/25
40000/40000 - 12s - loss: 0.0632 - acc: 0.9787 - val_loss: 1.5760 - val_acc:
0.7602
Epoch 22/25
40000/40000 - 12s - loss: 0.0571 - acc: 0.9801 - val_loss: 1.8291 - val_acc:
0.7553
Epoch 23/25
40000/40000 - 12s - loss: 0.0610 - acc: 0.9792 - val_loss: 2.0710 - val_acc:
0.7351
Epoch 24/25
40000/40000 - 12s - loss: 0.0545 - acc: 0.9812 - val_loss: 1.9122 - val_acc:
0.7548
Epoch 25/25
40000/40000 - 13s - loss: 0.0533 - acc: 0.9824 - val_loss: 1.8432 - val_acc:
0.7561
```

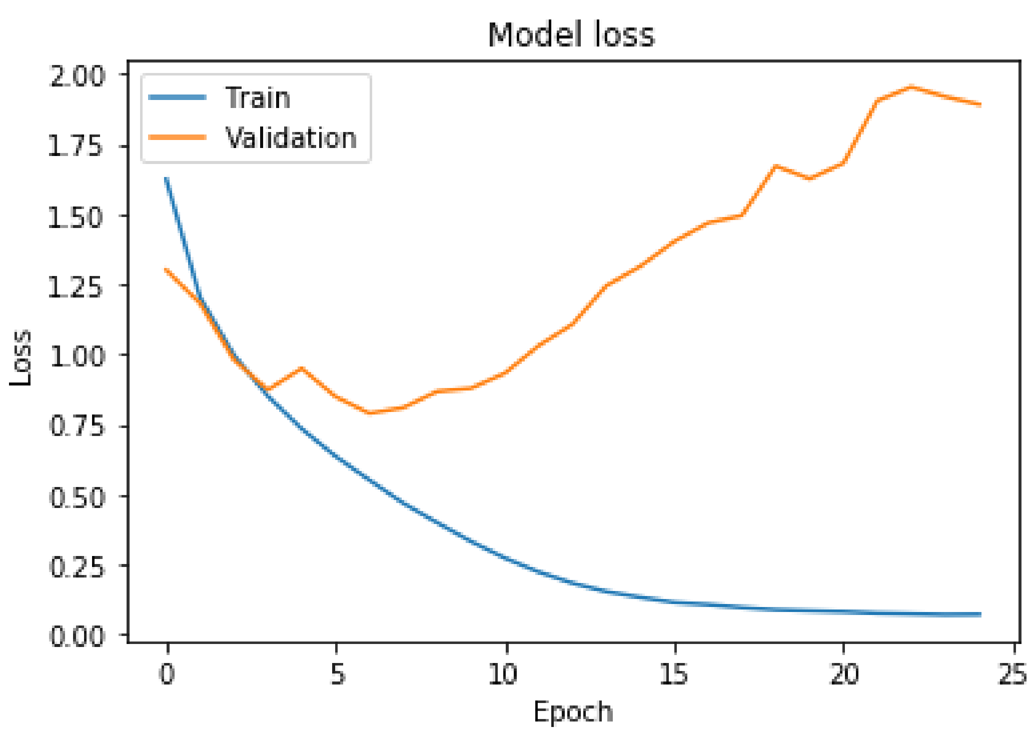
1.5.2 Validation

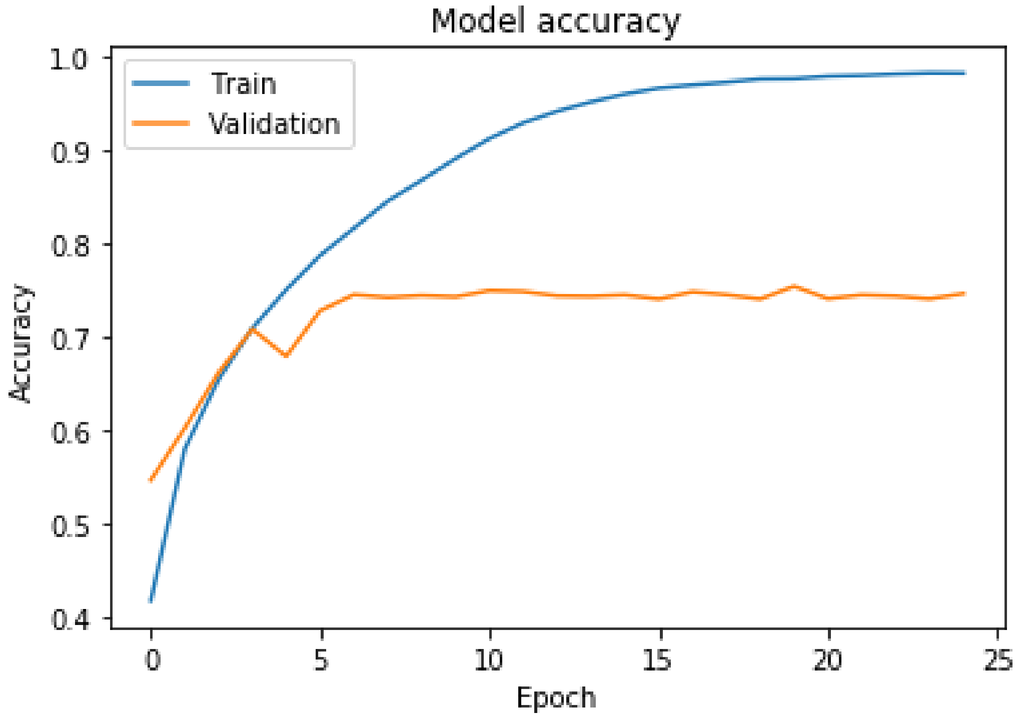
[0]:
```
print("Weight decay default 0.0005")
plot_graphs(history1)

print("Weight decay default 0.00005")
plot_graphs(history2)

print("Weight decay default 0.05")
plot_graphs(history3)

print("Weight decay default 0")
plot_graphs(history4)
```

```
Weight decay default 0.0005
```

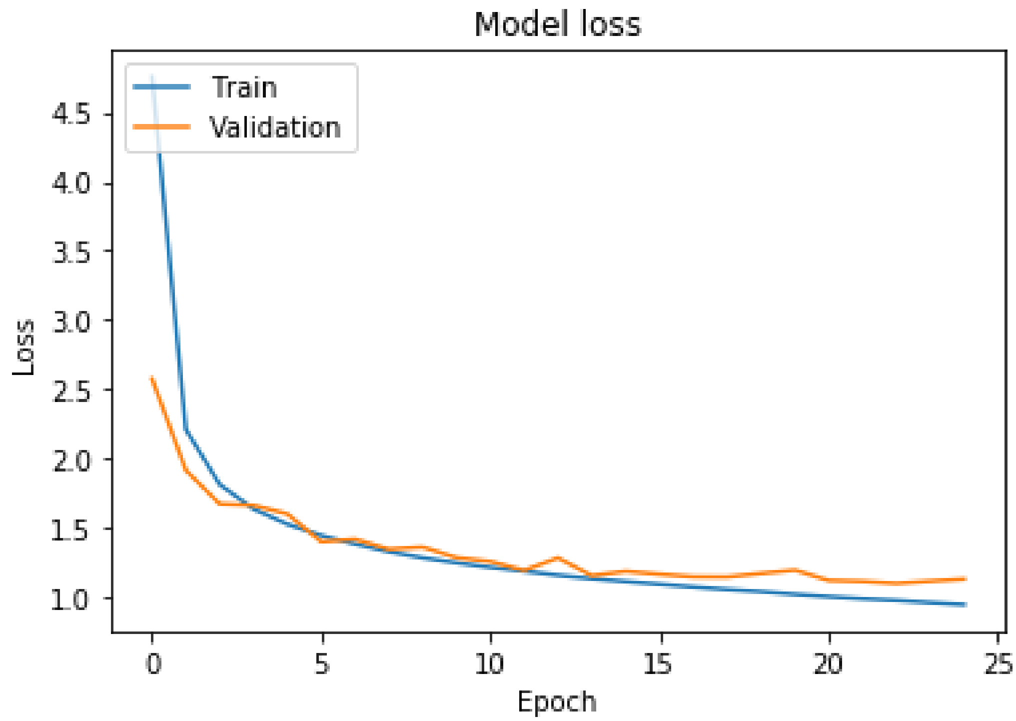


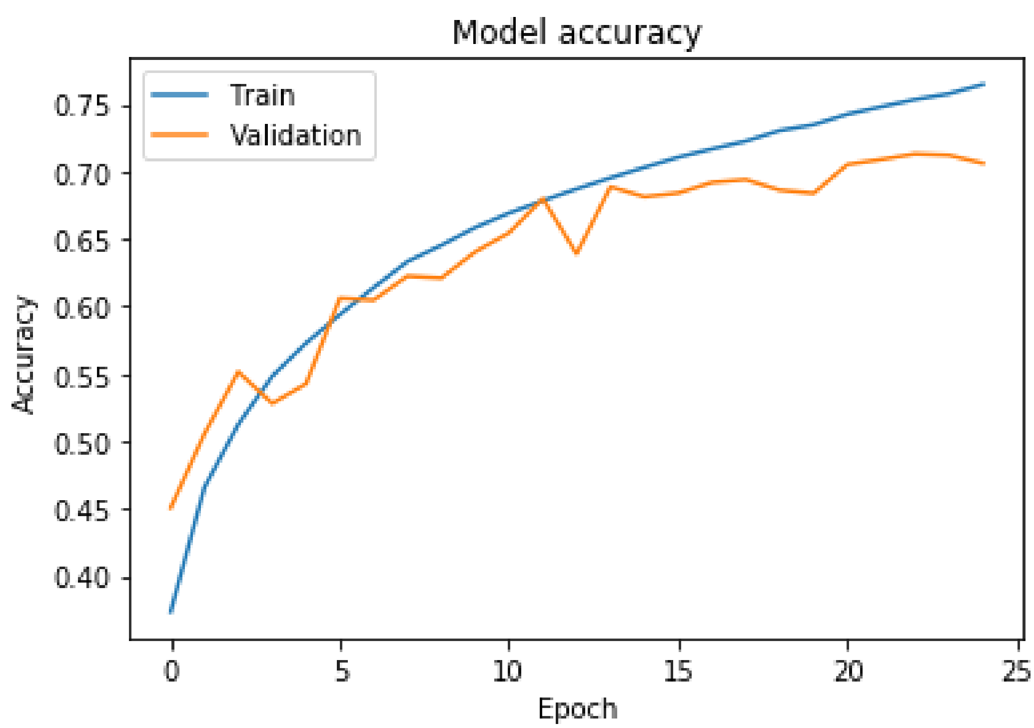
Weight decay default 0.00005



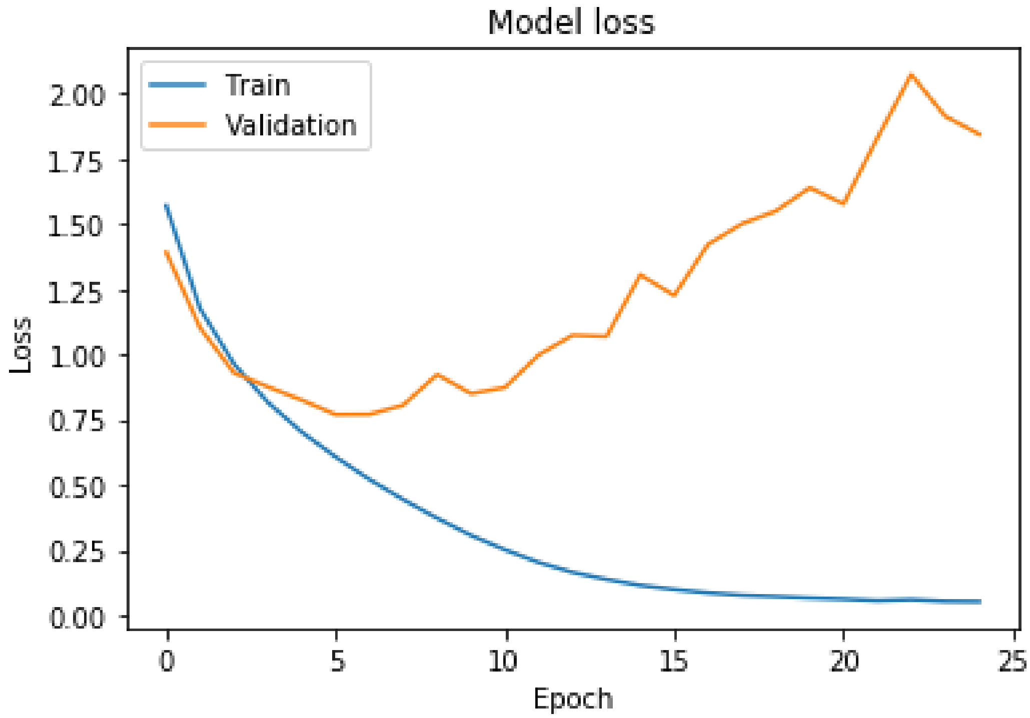


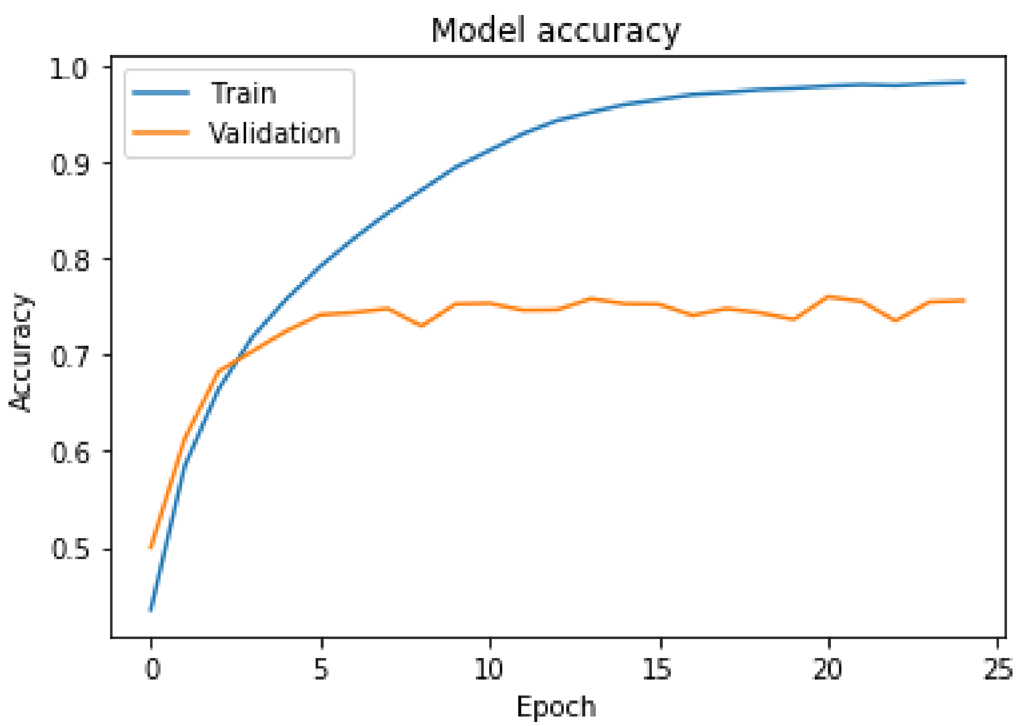

```
Weight decay default 0.05
```


Model loss
Train
Validation
Loss
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
5
10
15
20
25
Epoch


Model accuracy
Train
Validation
Accuracy
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0
5
10
15
20
25
Epoch

Weight decay default 0





As we can see by the graphs above, weight decay of 0.05 does well to reduce the difference between the training and validation model loss. However, it sacrifices some accuracy. The weight decay of 0.0005 has a relatively good accuracy but has a large gap between the training and validation model loss (overfitting). We shall proceed with the 0.05 weight decay or combine the 0.0005 weight decay with another regularization to further prevent overfitting.

1.6 Model #3 - Regularization with Dropout Layer

We will try adding a dropout layer of 0.25 for each step. We will also try a dropout of 0.5 as well a model with increasing levels of dropout.

1.6.1 Dropout 0.25

[0]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 input_shape=shape))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
```

```python
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compile the model
opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history1 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_3_part1.h5', overwrite=True)
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/25
40000/40000 - 20s - loss: 1.7901 - acc: 0.3407 - val_loss: 1.5220 - val_acc:
0.4631
Epoch 2/25
40000/40000 - 13s - loss: 1.4017 - acc: 0.4910 - val_loss: 1.1977 - val_acc:
0.5840
Epoch 3/25
40000/40000 - 13s - loss: 1.2017 - acc: 0.5708 - val_loss: 1.0828 - val_acc:
0.6257
Epoch 4/25
40000/40000 - 13s - loss: 1.0708 - acc: 0.6209 - val_loss: 0.9911 - val_acc:
0.6521
Epoch 5/25
40000/40000 - 13s - loss: 0.9753 - acc: 0.6580 - val_loss: 0.8944 - val_acc:
0.6891
Epoch 6/25
40000/40000 - 13s - loss: 0.8979 - acc: 0.6852 - val_loss: 0.8760 - val_acc:
0.7004
Epoch 7/25
```

```
40000/40000 - 13s - loss: 0.8461 - acc: 0.7045 - val_loss: 0.8249 - val_acc:
0.7214
Epoch 8/25
40000/40000 - 13s - loss: 0.8025 - acc: 0.7218 - val_loss: 0.7604 - val_acc:
0.7410
Epoch 9/25
40000/40000 - 13s - loss: 0.7602 - acc: 0.7364 - val_loss: 0.7194 - val_acc:
0.7532
Epoch 10/25
40000/40000 - 13s - loss: 0.7308 - acc: 0.7475 - val_loss: 0.7207 - val_acc:
0.7509
Epoch 11/25
40000/40000 - 13s - loss: 0.6998 - acc: 0.7584 - val_loss: 0.6691 - val_acc:
0.7730
Epoch 12/25
40000/40000 - 13s - loss: 0.6758 - acc: 0.7655 - val_loss: 0.6590 - val_acc:
0.7732
Epoch 13/25
40000/40000 - 13s - loss: 0.6483 - acc: 0.7740 - val_loss: 0.6252 - val_acc:
0.7854
Epoch 14/25
40000/40000 - 13s - loss: 0.6339 - acc: 0.7804 - val_loss: 0.6528 - val_acc:
0.7794
Epoch 15/25
40000/40000 - 13s - loss: 0.6114 - acc: 0.7871 - val_loss: 0.6334 - val_acc:
0.7873
Epoch 16/25
40000/40000 - 13s - loss: 0.5966 - acc: 0.7945 - val_loss: 0.6137 - val_acc:
0.7920
Epoch 17/25
40000/40000 - 13s - loss: 0.5877 - acc: 0.7992 - val_loss: 0.6050 - val_acc:
0.7923
Epoch 18/25
40000/40000 - 13s - loss: 0.5687 - acc: 0.8051 - val_loss: 0.6249 - val_acc:
0.7930
Epoch 19/25
40000/40000 - 13s - loss: 0.5536 - acc: 0.8087 - val_loss: 0.6179 - val_acc:
0.7929
Epoch 20/25
40000/40000 - 13s - loss: 0.5498 - acc: 0.8114 - val_loss: 0.5928 - val_acc:
0.8014
Epoch 21/25
40000/40000 - 13s - loss: 0.5400 - acc: 0.8141 - val_loss: 0.5971 - val_acc:
0.8003
Epoch 22/25
40000/40000 - 13s - loss: 0.5340 - acc: 0.8179 - val_loss: 0.5911 - val_acc:
0.7992
Epoch 23/25
```

```
40000/40000 - 13s - loss: 0.5221 - acc: 0.8217 - val_loss: 0.6040 - val_acc:
0.7976
Epoch 24/25
40000/40000 - 13s - loss: 0.5218 - acc: 0.8241 - val_loss: 0.5686 - val_acc:
0.8103
Epoch 25/25
40000/40000 - 13s - loss: 0.5162 - acc: 0.8235 - val_loss: 0.6255 - val_acc:
0.7918
```

1.6.2 Dropout 0.5

[0]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 input_shape=shape))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(256))
```

```
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compile the model
opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history2 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_3_part2.h5', overwrite=True)
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/25
40000/40000 - 20s - loss: 2.0588 - acc: 0.2271 - val_loss: 1.7303 - val_acc:
0.3730
Epoch 2/25
40000/40000 - 13s - loss: 1.6801 - acc: 0.3812 - val_loss: 1.6171 - val_acc:
0.4129
Epoch 3/25
40000/40000 - 13s - loss: 1.5185 - acc: 0.4439 - val_loss: 1.4503 - val_acc:
0.4793
Epoch 4/25
40000/40000 - 13s - loss: 1.4118 - acc: 0.4872 - val_loss: 1.2691 - val_acc:
0.5562
Epoch 5/25
40000/40000 - 13s - loss: 1.3326 - acc: 0.5207 - val_loss: 1.2628 - val_acc:
0.5499
Epoch 6/25
40000/40000 - 13s - loss: 1.2555 - acc: 0.5507 - val_loss: 1.1990 - val_acc:
0.5720
Epoch 7/25
40000/40000 - 13s - loss: 1.1948 - acc: 0.5756 - val_loss: 1.1330 - val_acc:
0.5991
Epoch 8/25
40000/40000 - 13s - loss: 1.1514 - acc: 0.5938 - val_loss: 1.0354 - val_acc:
0.6338
```

```
Epoch 9/25
40000/40000 - 13s - loss: 1.1173 - acc: 0.6089 - val_loss: 1.0001 - val_acc:
0.6501
Epoch 10/25
40000/40000 - 13s - loss: 1.0780 - acc: 0.6209 - val_loss: 1.0033 - val_acc:
0.6409
Epoch 11/25
40000/40000 - 13s - loss: 1.0516 - acc: 0.6295 - val_loss: 0.9677 - val_acc:
0.6583
Epoch 12/25
40000/40000 - 13s - loss: 1.0153 - acc: 0.6444 - val_loss: 0.9746 - val_acc:
0.6578
Epoch 13/25
40000/40000 - 13s - loss: 0.9984 - acc: 0.6492 - val_loss: 0.8731 - val_acc:
0.6953
Epoch 14/25
40000/40000 - 13s - loss: 0.9771 - acc: 0.6582 - val_loss: 0.9261 - val_acc:
0.6743
Epoch 15/25
40000/40000 - 13s - loss: 0.9666 - acc: 0.6617 - val_loss: 0.8705 - val_acc:
0.6893
Epoch 16/25
40000/40000 - 13s - loss: 0.9472 - acc: 0.6705 - val_loss: 0.8185 - val_acc:
0.7146
Epoch 17/25
40000/40000 - 13s - loss: 0.9359 - acc: 0.6771 - val_loss: 0.8238 - val_acc:
0.7174
Epoch 18/25
40000/40000 - 13s - loss: 0.9267 - acc: 0.6781 - val_loss: 0.8047 - val_acc:
0.7174
Epoch 19/25
40000/40000 - 13s - loss: 0.9149 - acc: 0.6809 - val_loss: 0.8146 - val_acc:
0.7218
Epoch 20/25
40000/40000 - 13s - loss: 0.9016 - acc: 0.6903 - val_loss: 0.8249 - val_acc:
0.7252
Epoch 21/25
40000/40000 - 13s - loss: 0.8949 - acc: 0.6902 - val_loss: 0.8127 - val_acc:
0.7278
Epoch 22/25
40000/40000 - 13s - loss: 0.8855 - acc: 0.6947 - val_loss: 0.8227 - val_acc:
0.7152
Epoch 23/25
40000/40000 - 13s - loss: 0.8846 - acc: 0.6941 - val_loss: 0.7816 - val_acc:
0.7389
Epoch 24/25
40000/40000 - 13s - loss: 0.8712 - acc: 0.7028 - val_loss: 0.7798 - val_acc:
0.7307
```

```
Epoch 25/25
40000/40000 - 13s - loss: 0.8723 - acc: 0.6981 - val_loss: 0.8437 - val_acc:
0.7122
```

1.6.3 Dropout increasing from 0.15 to 0.45

[0]:
```python
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 input_shape=shape))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.15))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.45))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

```python
# Compile the model
opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history3 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_3_part3.h5', overwrite=True)
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/25
40000/40000 - 20s - loss: 1.8625 - acc: 0.3076 - val_loss: 1.5225 - val_acc:
0.4586
Epoch 2/25
40000/40000 - 13s - loss: 1.4921 - acc: 0.4582 - val_loss: 1.3287 - val_acc:
0.5293
Epoch 3/25
40000/40000 - 13s - loss: 1.3044 - acc: 0.5333 - val_loss: 1.1574 - val_acc:
0.5823
Epoch 4/25
40000/40000 - 13s - loss: 1.1641 - acc: 0.5888 - val_loss: 1.0121 - val_acc:
0.6407
Epoch 5/25
40000/40000 - 13s - loss: 1.0642 - acc: 0.6251 - val_loss: 0.9195 - val_acc:
0.6829
Epoch 6/25
40000/40000 - 13s - loss: 0.9907 - acc: 0.6522 - val_loss: 0.8662 - val_acc:
0.7002
Epoch 7/25
40000/40000 - 13s - loss: 0.9320 - acc: 0.6741 - val_loss: 0.7970 - val_acc:
0.7180
Epoch 8/25
40000/40000 - 13s - loss: 0.8830 - acc: 0.6923 - val_loss: 0.7806 - val_acc:
0.7291
Epoch 9/25
40000/40000 - 13s - loss: 0.8439 - acc: 0.7070 - val_loss: 0.7376 - val_acc:
0.7429
Epoch 10/25
40000/40000 - 13s - loss: 0.8051 - acc: 0.7204 - val_loss: 0.7162 - val_acc:
```

```
0.7527
Epoch 11/25
40000/40000 - 13s - loss: 0.7763 - acc: 0.7310 - val_loss: 0.7041 - val_acc:
0.7589
Epoch 12/25
40000/40000 - 13s - loss: 0.7472 - acc: 0.7415 - val_loss: 0.7282 - val_acc:
0.7493
Epoch 13/25
40000/40000 - 13s - loss: 0.7234 - acc: 0.7516 - val_loss: 0.6809 - val_acc:
0.7650
Epoch 14/25
40000/40000 - 13s - loss: 0.7038 - acc: 0.7578 - val_loss: 0.6750 - val_acc:
0.7704
Epoch 15/25
40000/40000 - 13s - loss: 0.6895 - acc: 0.7637 - val_loss: 0.7180 - val_acc:
0.7520
Epoch 16/25
40000/40000 - 13s - loss: 0.6686 - acc: 0.7713 - val_loss: 0.6482 - val_acc:
0.7779
Epoch 17/25
40000/40000 - 13s - loss: 0.6540 - acc: 0.7771 - val_loss: 0.6316 - val_acc:
0.7806
Epoch 18/25
40000/40000 - 13s - loss: 0.6378 - acc: 0.7799 - val_loss: 0.6279 - val_acc:
0.7898
Epoch 19/25
40000/40000 - 13s - loss: 0.6244 - acc: 0.7867 - val_loss: 0.6171 - val_acc:
0.7898
Epoch 20/25
40000/40000 - 13s - loss: 0.6169 - acc: 0.7904 - val_loss: 0.6156 - val_acc:
0.7912
Epoch 21/25
40000/40000 - 13s - loss: 0.6081 - acc: 0.7905 - val_loss: 0.6084 - val_acc:
0.7913
Epoch 22/25
40000/40000 - 13s - loss: 0.5988 - acc: 0.7970 - val_loss: 0.6255 - val_acc:
0.7883
Epoch 23/25
40000/40000 - 13s - loss: 0.5910 - acc: 0.7969 - val_loss: 0.5970 - val_acc:
0.8003
Epoch 24/25
40000/40000 - 13s - loss: 0.5802 - acc: 0.8039 - val_loss: 0.6012 - val_acc:
0.7959
Epoch 25/25
40000/40000 - 13s - loss: 0.5768 - acc: 0.8035 - val_loss: 0.6313 - val_acc:
0.7891
```

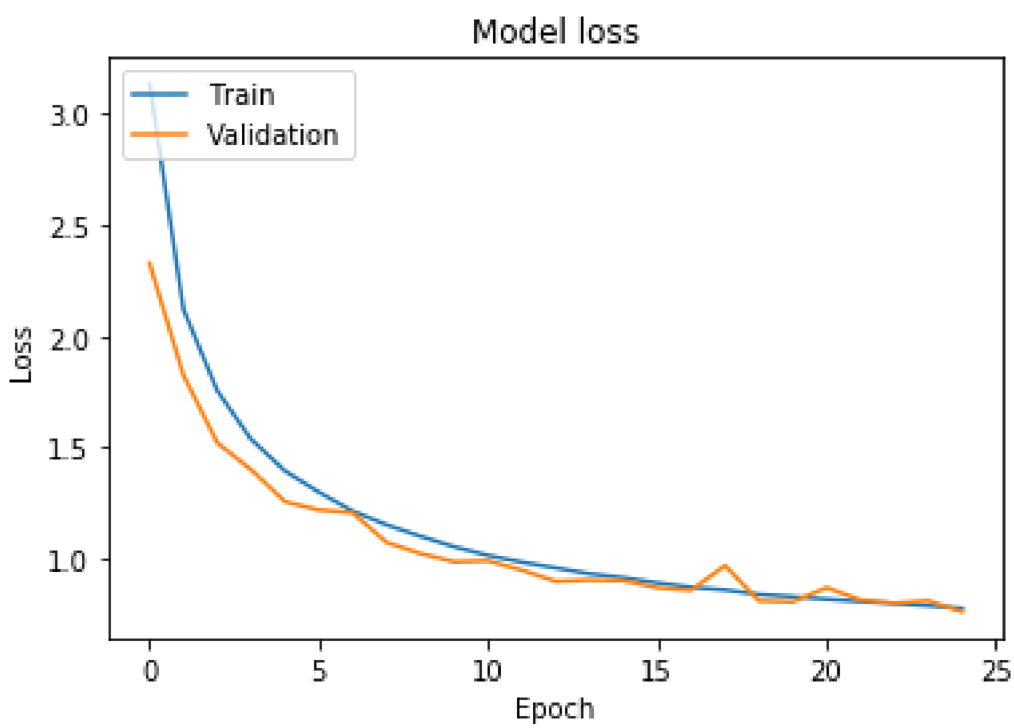
1.6.4 Validation

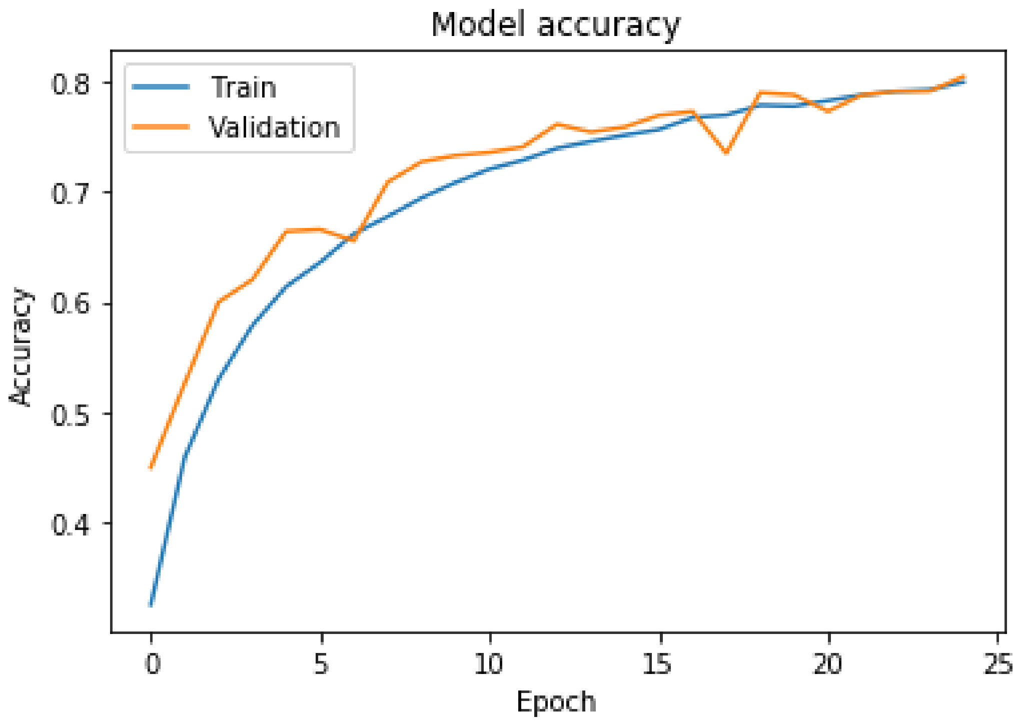
The dropout layer of 0.25 and increasing from 0.15 to 0.45 seem to be better at both increasing accuracy of the training and validation set as well as reducing the loss between the two. Potentially increasing the number of epochs can smooth the curves (we will try increasing the number of epochs later).

We will either use these dropout layers for regularization to prevent overfitting or we will combine the dropout layer and weight decay together as weight decay alone is much worse at preventing overfitting than the dropout layer.

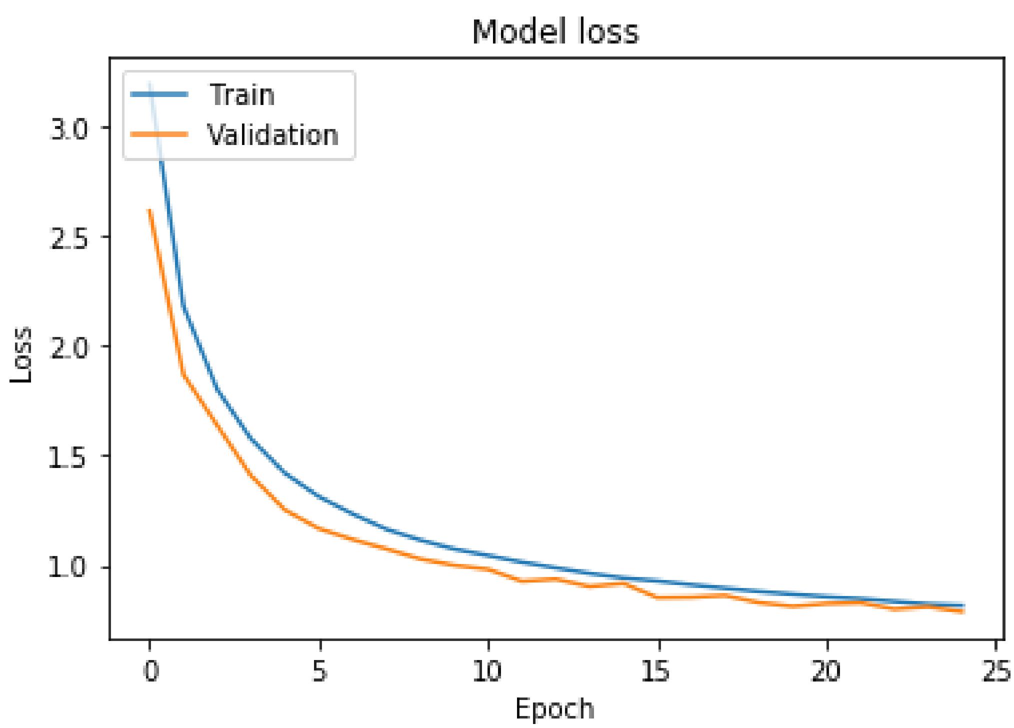
[0]:
```
print("Dropout 0.25")
plot_graphs(history1)

print("Dropout 0.5")
plot_graphs(history2)

print("Dropout increasing from 0.15 -> 0.45")
plot_graphs(history3)
```

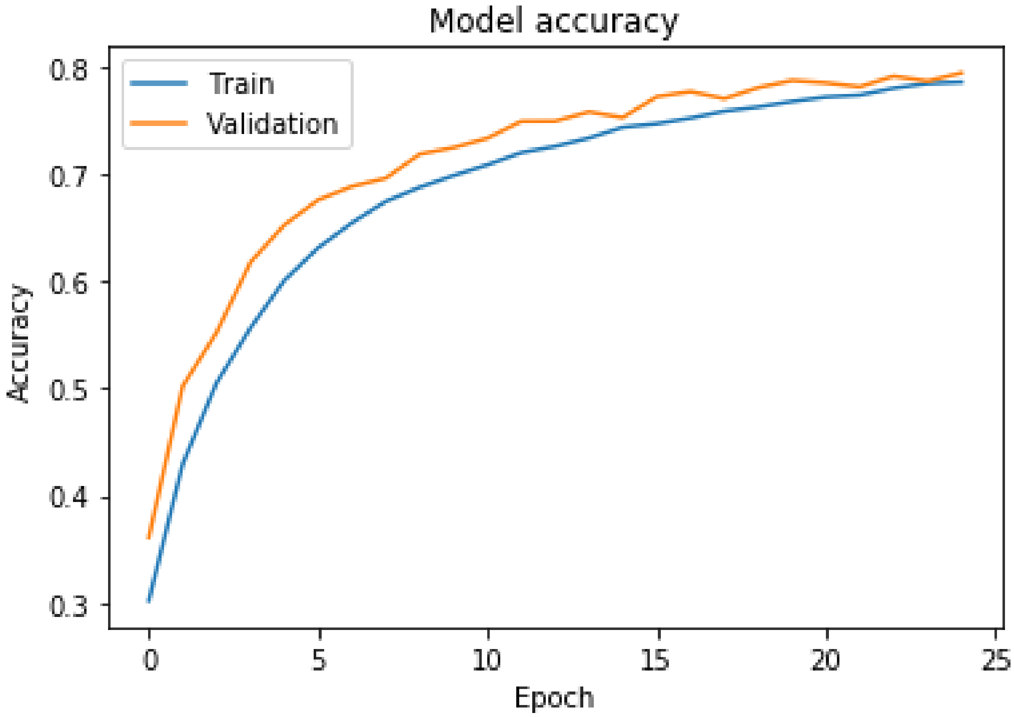
```
Dropout 0.25
```



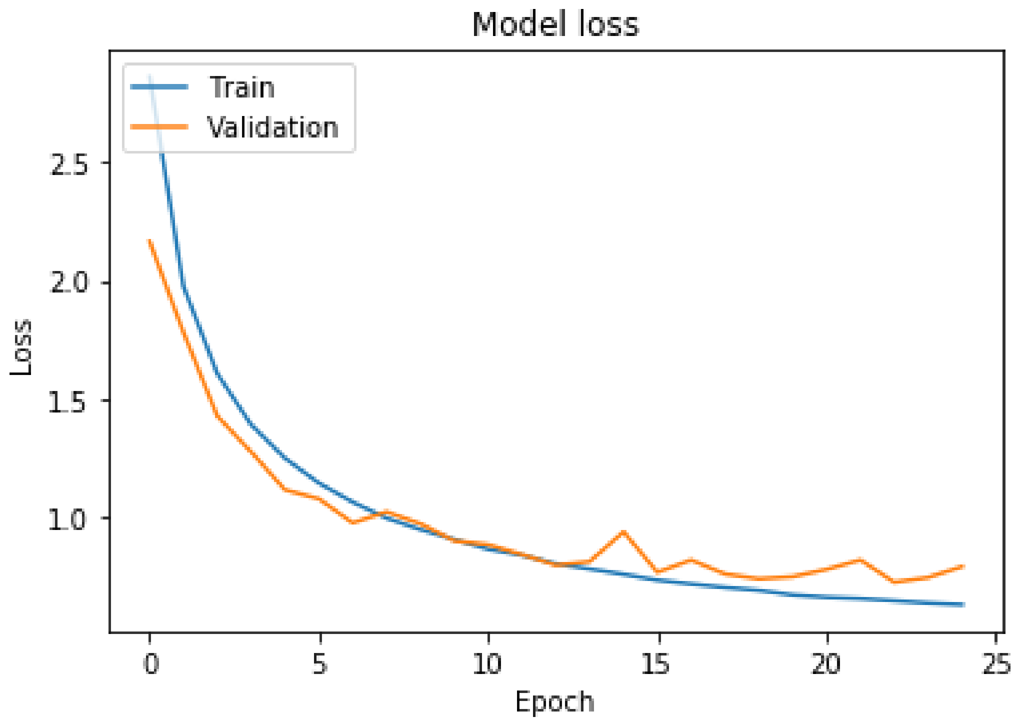

Model accuracy
Train
Validation
0.8
0.7
0.6
0.5
0.4
Accuracy
0
5
10
15
20
25
Epoch

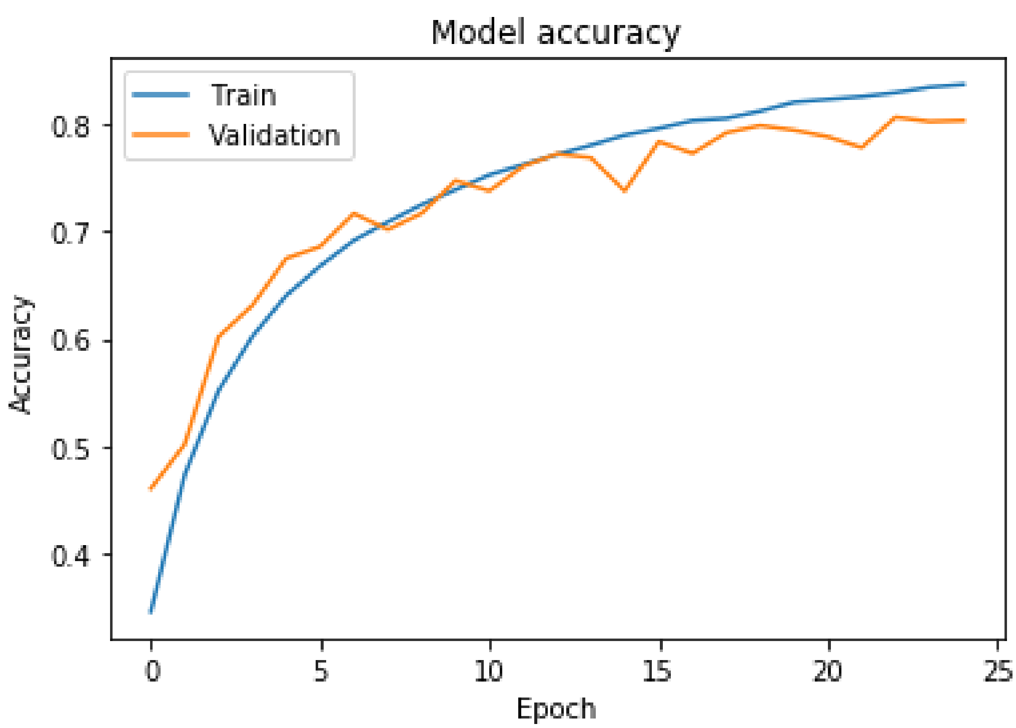
Dropout 0.5


Model loss
Train
Validation
3.0
2.5
2.0
1.5
1.0
Loss
0
5
10
15
20
25
Epoch



Dropout increasing from 0.15 -> 0.45


Model loss
Train
Validation
2.5
2.0
1.5
1.0
Loss
0
5
10
15
20
25
Epoch


Model accuracy
Train
Validation
0.8
0.7
0.6
0.5
0.4
Accuracy
0
5
10
15
20
25
Epoch

1.7 Model #4 - Regularization Dropout and Weight Decay

Now, we will try combining the dropout and weight decay regularizations together to see if they are better than our best dropout layers (0.25 and 0.15->0.45)

1.7.1 Dropout 0.25, weight decay 0.0005

[0]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(),
                 input_shape=shape))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
```

```python
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compile the model
opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history1 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_4_part1.h5', overwrite=True)
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/25
40000/40000 - 21s - loss: 3.1346 - acc: 0.3250 - val_loss: 2.3277 - val_acc:
0.4496
Epoch 2/25
40000/40000 - 13s - loss: 2.1222 - acc: 0.4586 - val_loss: 1.8253 - val_acc:
0.5261
Epoch 3/25
40000/40000 - 14s - loss: 1.7554 - acc: 0.5297 - val_loss: 1.5207 - val_acc:
0.5998
Epoch 4/25
40000/40000 - 13s - loss: 1.5361 - acc: 0.5784 - val_loss: 1.4004 - val_acc:
0.6203
Epoch 5/25
40000/40000 - 13s - loss: 1.3944 - acc: 0.6138 - val_loss: 1.2562 - val_acc:
0.6639
Epoch 6/25
40000/40000 - 13s - loss: 1.2991 - acc: 0.6358 - val_loss: 1.2189 - val_acc:
0.6656
Epoch 7/25
40000/40000 - 13s - loss: 1.2137 - acc: 0.6616 - val_loss: 1.2066 - val_acc:
0.6554
Epoch 8/25
40000/40000 - 13s - loss: 1.1523 - acc: 0.6775 - val_loss: 1.0730 - val_acc:
```

```
0.7088
Epoch 9/25
40000/40000 - 13s - loss: 1.1008 - acc: 0.6944 - val_loss: 1.0228 - val_acc:
0.7273
Epoch 10/25
40000/40000 - 13s - loss: 1.0529 - acc: 0.7084 - val_loss: 0.9856 - val_acc:
0.7329
Epoch 11/25
40000/40000 - 13s - loss: 1.0141 - acc: 0.7206 - val_loss: 0.9902 - val_acc:
0.7358
Epoch 12/25
40000/40000 - 13s - loss: 0.9839 - acc: 0.7287 - val_loss: 0.9479 - val_acc:
0.7408
Epoch 13/25
40000/40000 - 13s - loss: 0.9580 - acc: 0.7395 - val_loss: 0.8976 - val_acc:
0.7612
Epoch 14/25
40000/40000 - 14s - loss: 0.9316 - acc: 0.7456 - val_loss: 0.9037 - val_acc:
0.7542
Epoch 15/25
40000/40000 - 14s - loss: 0.9139 - acc: 0.7513 - val_loss: 0.9007 - val_acc:
0.7587
Epoch 16/25
40000/40000 - 13s - loss: 0.8906 - acc: 0.7563 - val_loss: 0.8689 - val_acc:
0.7693
Epoch 17/25
40000/40000 - 13s - loss: 0.8709 - acc: 0.7672 - val_loss: 0.8576 - val_acc:
0.7728
Epoch 18/25
40000/40000 - 13s - loss: 0.8589 - acc: 0.7696 - val_loss: 0.9687 - val_acc:
0.7350
Epoch 19/25
40000/40000 - 13s - loss: 0.8386 - acc: 0.7785 - val_loss: 0.8098 - val_acc:
0.7900
Epoch 20/25
40000/40000 - 14s - loss: 0.8283 - acc: 0.7777 - val_loss: 0.8056 - val_acc:
0.7880
Epoch 21/25
40000/40000 - 13s - loss: 0.8172 - acc: 0.7827 - val_loss: 0.8689 - val_acc:
0.7730
Epoch 22/25
40000/40000 - 13s - loss: 0.8071 - acc: 0.7879 - val_loss: 0.8140 - val_acc:
0.7878
Epoch 23/25
40000/40000 - 13s - loss: 0.7980 - acc: 0.7911 - val_loss: 0.7990 - val_acc:
0.7913
Epoch 24/25
40000/40000 - 13s - loss: 0.7880 - acc: 0.7925 - val_loss: 0.8101 - val_acc:
```

```
0.7913
Epoch 25/25
40000/40000 - 13s - loss: 0.7763 - acc: 0.7996 - val_loss: 0.7612 - val_acc:
0.8045
```

1.7.2 Dropout increasing, weight decay 0.0005

[0]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(),
                 input_shape=shape))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.15))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35))
```

```
model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.45))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compile the model
opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history2 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_4_part2.h5', overwrite=True)
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/25
40000/40000 - 21s - loss: 3.1879 - acc: 0.3022 - val_loss: 2.6121 - val_acc:
0.3612
Epoch 2/25
40000/40000 - 13s - loss: 2.1792 - acc: 0.4294 - val_loss: 1.8680 - val_acc:
0.5020
Epoch 3/25
40000/40000 - 13s - loss: 1.8000 - acc: 0.5050 - val_loss: 1.6364 - val_acc:
0.5520
Epoch 4/25
40000/40000 - 13s - loss: 1.5745 - acc: 0.5561 - val_loss: 1.4080 - val_acc:
0.6173
Epoch 5/25
40000/40000 - 13s - loss: 1.4192 - acc: 0.6004 - val_loss: 1.2522 - val_acc:
0.6520
Epoch 6/25
40000/40000 - 13s - loss: 1.3133 - acc: 0.6307 - val_loss: 1.1678 - val_acc:
0.6755
Epoch 7/25
40000/40000 - 13s - loss: 1.2342 - acc: 0.6541 - val_loss: 1.1183 - val_acc:
0.6881
Epoch 8/25
```

```
40000/40000 - 13s - loss: 1.1647 - acc: 0.6741 - val_loss: 1.0753 - val_acc:
0.6958
Epoch 9/25
40000/40000 - 13s - loss: 1.1156 - acc: 0.6873 - val_loss: 1.0295 - val_acc:
0.7182
Epoch 10/25
40000/40000 - 13s - loss: 1.0747 - acc: 0.6982 - val_loss: 1.0007 - val_acc:
0.7243
Epoch 11/25
40000/40000 - 13s - loss: 1.0451 - acc: 0.7082 - val_loss: 0.9828 - val_acc:
0.7330
Epoch 12/25
40000/40000 - 13s - loss: 1.0151 - acc: 0.7196 - val_loss: 0.9283 - val_acc:
0.7488
Epoch 13/25
40000/40000 - 13s - loss: 0.9891 - acc: 0.7254 - val_loss: 0.9379 - val_acc:
0.7489
Epoch 14/25
40000/40000 - 13s - loss: 0.9642 - acc: 0.7332 - val_loss: 0.9045 - val_acc:
0.7576
Epoch 15/25
40000/40000 - 13s - loss: 0.9430 - acc: 0.7432 - val_loss: 0.9185 - val_acc:
0.7523
Epoch 16/25
40000/40000 - 13s - loss: 0.9294 - acc: 0.7466 - val_loss: 0.8538 - val_acc:
0.7718
Epoch 17/25
40000/40000 - 13s - loss: 0.9123 - acc: 0.7518 - val_loss: 0.8556 - val_acc:
0.7765
Epoch 18/25
40000/40000 - 13s - loss: 0.8966 - acc: 0.7582 - val_loss: 0.8632 - val_acc:
0.7700
Epoch 19/25
40000/40000 - 13s - loss: 0.8817 - acc: 0.7622 - val_loss: 0.8311 - val_acc:
0.7802
Epoch 20/25
40000/40000 - 13s - loss: 0.8690 - acc: 0.7673 - val_loss: 0.8142 - val_acc:
0.7868
Epoch 21/25
40000/40000 - 13s - loss: 0.8574 - acc: 0.7715 - val_loss: 0.8266 - val_acc:
0.7846
Epoch 22/25
40000/40000 - 13s - loss: 0.8488 - acc: 0.7732 - val_loss: 0.8306 - val_acc:
0.7810
Epoch 23/25
40000/40000 - 13s - loss: 0.8364 - acc: 0.7796 - val_loss: 0.8020 - val_acc:
0.7907
Epoch 24/25
```

```
40000/40000 - 13s - loss: 0.8237 - acc: 0.7841 - val_loss: 0.8122 - val_acc:
0.7862
Epoch 25/25
40000/40000 - 13s - loss: 0.8182 - acc: 0.7853 - val_loss: 0.7901 - val_acc:
0.7937
```

1.7.3 Dropout increasing, weight decay 0.0005, both alternating

[0]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(),
                 input_shape=shape))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                  kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
```

```
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compile the model
opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history3 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_4_part3.h5', overwrite=True)
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/25
40000/40000 - 21s - loss: 2.8603 - acc: 0.3458 - val_loss: 2.1650 - val_acc:
0.4605
Epoch 2/25
40000/40000 - 13s - loss: 1.9755 - acc: 0.4733 - val_loss: 1.7810 - val_acc:
0.5016
Epoch 3/25
40000/40000 - 13s - loss: 1.6072 - acc: 0.5516 - val_loss: 1.4278 - val_acc:
0.6018
Epoch 4/25
40000/40000 - 13s - loss: 1.3911 - acc: 0.6022 - val_loss: 1.2760 - val_acc:
0.6311
Epoch 5/25
40000/40000 - 13s - loss: 1.2491 - acc: 0.6403 - val_loss: 1.1161 - val_acc:
0.6747
Epoch 6/25
40000/40000 - 13s - loss: 1.1450 - acc: 0.6676 - val_loss: 1.0785 - val_acc:
0.6857
Epoch 7/25
40000/40000 - 13s - loss: 1.0644 - acc: 0.6915 - val_loss: 0.9764 - val_acc:
0.7166
Epoch 8/25
40000/40000 - 13s - loss: 0.9969 - acc: 0.7087 - val_loss: 1.0232 - val_acc:
0.7016
Epoch 9/25
```

```
40000/40000 - 13s - loss: 0.9486 - acc: 0.7249 - val_loss: 0.9733 - val_acc:
0.7167
Epoch 10/25
40000/40000 - 13s - loss: 0.9062 - acc: 0.7387 - val_loss: 0.9004 - val_acc:
0.7472
Epoch 11/25
40000/40000 - 13s - loss: 0.8656 - acc: 0.7526 - val_loss: 0.8845 - val_acc:
0.7377
Epoch 12/25
40000/40000 - 13s - loss: 0.8399 - acc: 0.7622 - val_loss: 0.8427 - val_acc:
0.7606
Epoch 13/25
40000/40000 - 13s - loss: 0.8040 - acc: 0.7715 - val_loss: 0.7979 - val_acc:
0.7722
Epoch 14/25
40000/40000 - 13s - loss: 0.7818 - acc: 0.7802 - val_loss: 0.8128 - val_acc:
0.7686
Epoch 15/25
40000/40000 - 13s - loss: 0.7593 - acc: 0.7896 - val_loss: 0.9388 - val_acc:
0.7376
Epoch 16/25
40000/40000 - 13s - loss: 0.7350 - acc: 0.7959 - val_loss: 0.7680 - val_acc:
0.7836
Epoch 17/25
40000/40000 - 13s - loss: 0.7188 - acc: 0.8031 - val_loss: 0.8194 - val_acc:
0.7727
Epoch 18/25
40000/40000 - 13s - loss: 0.7043 - acc: 0.8053 - val_loss: 0.7605 - val_acc:
0.7919
Epoch 19/25
40000/40000 - 13s - loss: 0.6919 - acc: 0.8120 - val_loss: 0.7407 - val_acc:
0.7986
Epoch 20/25
40000/40000 - 13s - loss: 0.6725 - acc: 0.8203 - val_loss: 0.7495 - val_acc:
0.7943
Epoch 21/25
40000/40000 - 13s - loss: 0.6624 - acc: 0.8230 - val_loss: 0.7800 - val_acc:
0.7881
Epoch 22/25
40000/40000 - 13s - loss: 0.6572 - acc: 0.8255 - val_loss: 0.8197 - val_acc:
0.7781
Epoch 23/25
40000/40000 - 13s - loss: 0.6480 - acc: 0.8292 - val_loss: 0.7262 - val_acc:
0.8064
Epoch 24/25
40000/40000 - 13s - loss: 0.6385 - acc: 0.8342 - val_loss: 0.7461 - val_acc:
0.8023
Epoch 25/25
```

```
40000/40000 - 13s - loss: 0.6316 - acc: 0.8367 - val_loss: 0.7916 - val_acc:
0.8032
```

1.7.4 Validation

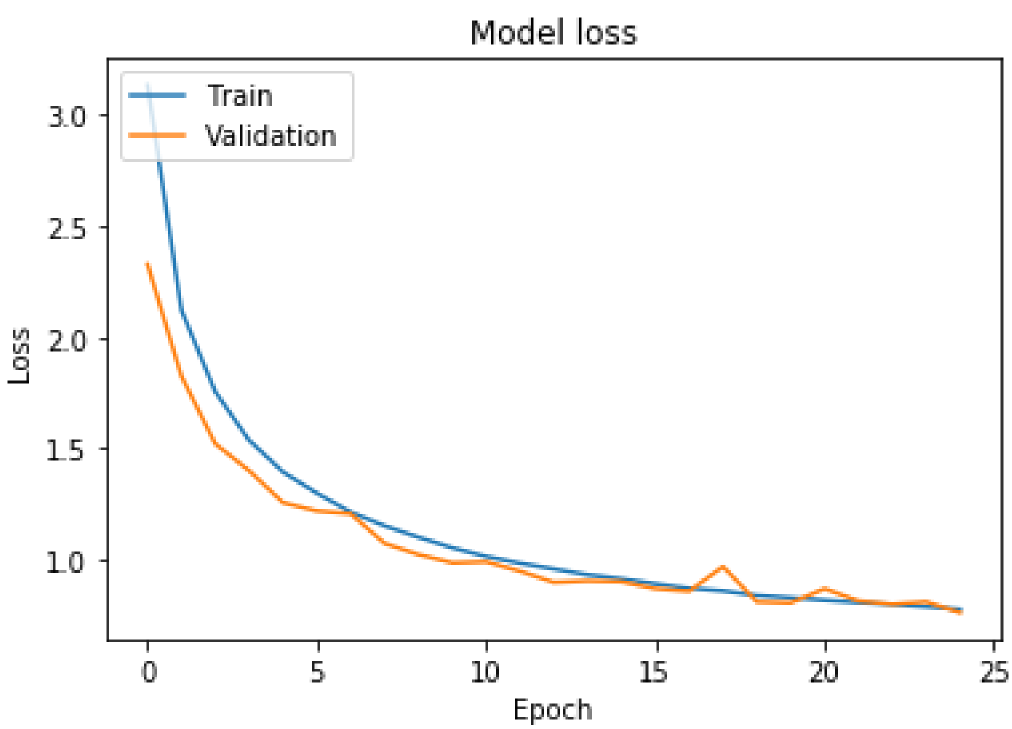
From the graphs below, it seems like alternating increasing dropout from 0.15 to 0.45 and and weight decay of 0.0005 increases accuracy and minimizes the training and validation loss the most. However, just increasing the dropout 0.15 to 0.45 produced similar accuracies but minimized the training and validation loss moreso than the combination of dropout and weight decay.

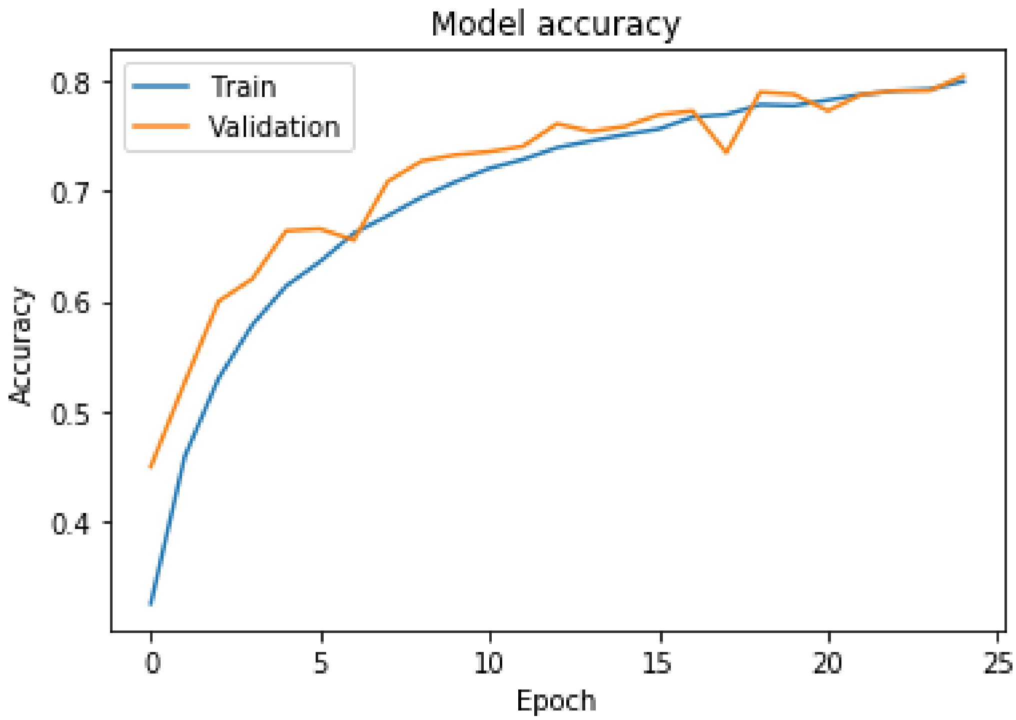
For the section below, we will proceed by using seeing how normalization impacts our best models so far.

1. increasing dropout from 0.15 to 0.45
2. increasing dropout from 0.15 to 0.45 and and weight decay of 0.0005)

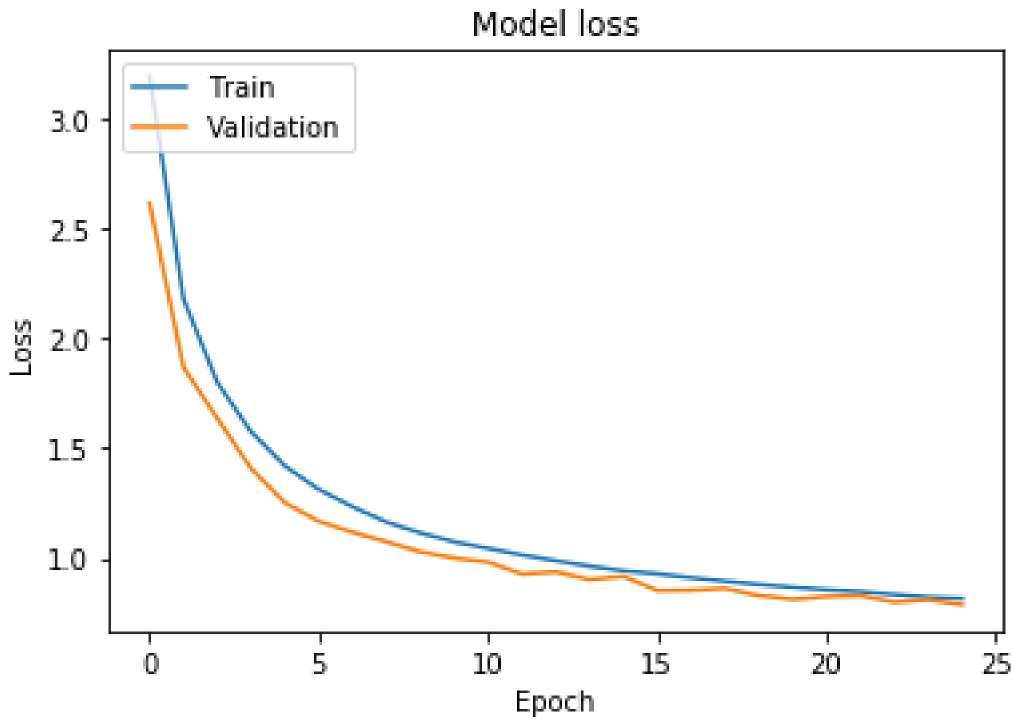
[0]:
```python
print("Dropout 0.25, weight decay 0.0005")
plot_graphs(history1)

print("Dropout increasing from 0.15 -> 0.45, weight decay 0.0005")
plot_graphs(history2)

print("Alternating dropout (increasing) and weight decay 0.0005")
plot_graphs(history3)
```

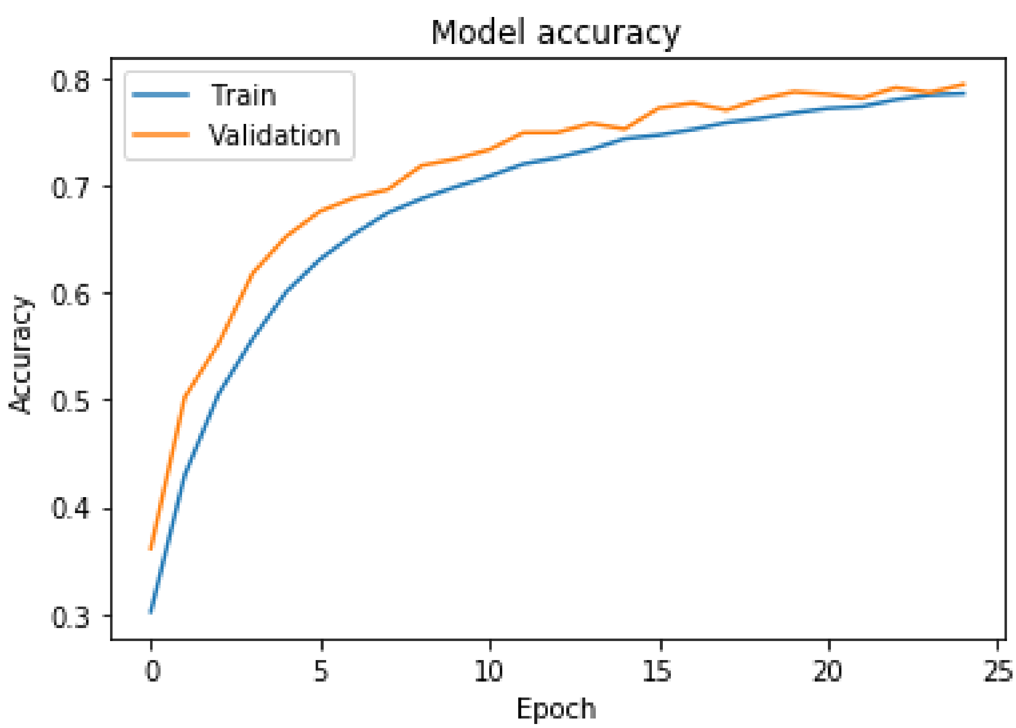
```
Dropout 0.25, weight decay 0.0005
```



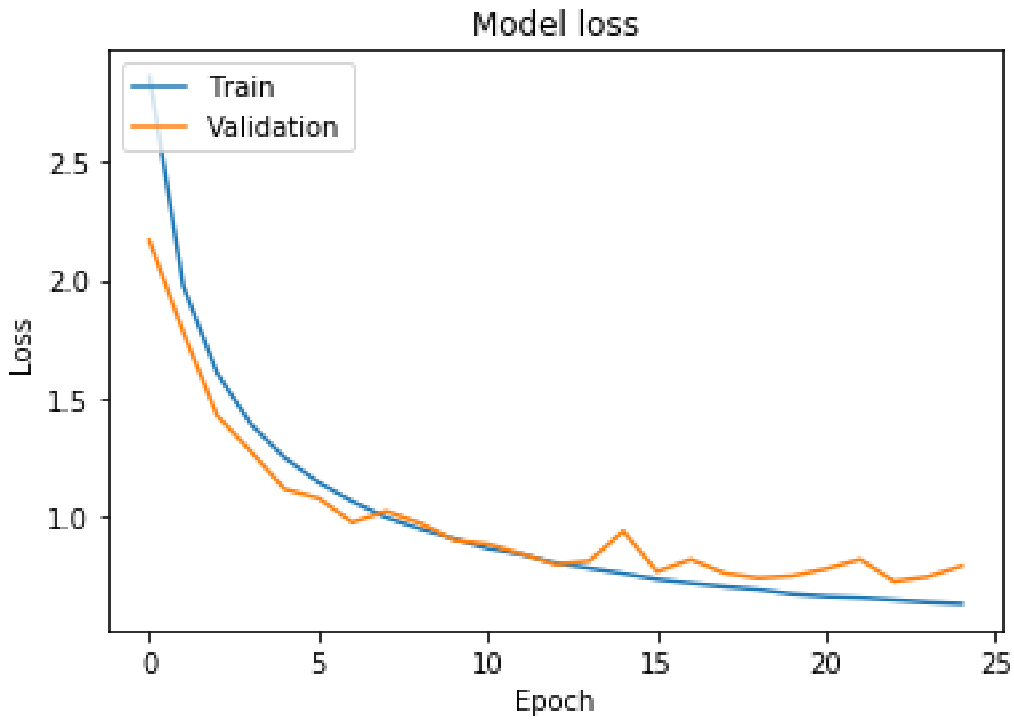


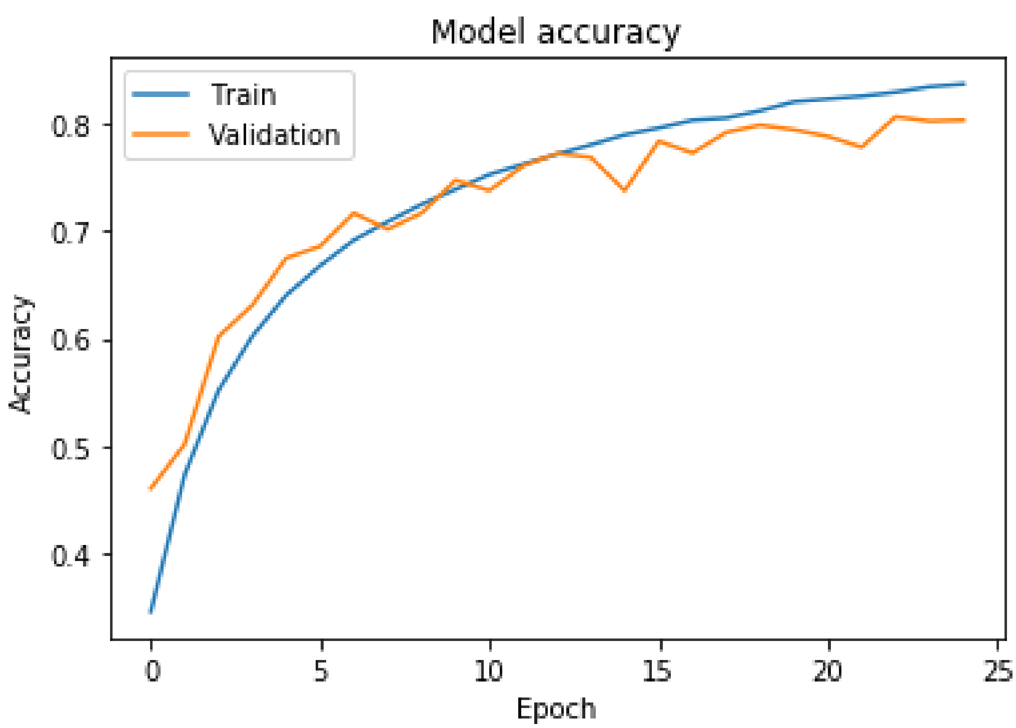
`Dropout increasing from 0.15 -> 0.45, weight decay 0.0005`


Model loss
Train
Validation
3.0
2.5
2.0
1.5
1.0
Loss
0
5
10
15
20
25
Epoch


Model accuracy
Train
Validation
0.8
0.7
0.6
0.5
0.4
0.3
Accuracy
0
5
10
15
20
25
Epoch

Alternating dropout (increasing) and weight decay 0.0005





1.8 Model #5 - Normalization

Now that we added regularization to the model, let's also add normalization. We will add Batch Normalization, which will normalize the activations of the previous hidden layer at each batch.

Let's see how regularizing and normalizing the layers compares to just regularizing.

1.8.1 Dropout increasing with normalization

[15]:
```
from tensorflow.keras.layers import BatchNormalization
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 input_shape=shape))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.15))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3),
                 padding='same',
```

```
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Dropout(0.45))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compile the model
opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history1 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_5_part1.h5', overwrite=True)
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/25
40000/40000 - 12s - loss: 2.2320 - acc: 0.3167 - val_loss: 1.5173 - val_acc:
0.4633
Epoch 2/25
40000/40000 - 11s - loss: 1.6244 - acc: 0.4555 - val_loss: 1.2026 - val_acc:
0.5707
Epoch 3/25
40000/40000 - 11s - loss: 1.3475 - acc: 0.5423 - val_loss: 1.0285 - val_acc:
0.6400
Epoch 4/25
40000/40000 - 11s - loss: 1.1582 - acc: 0.5988 - val_loss: 1.0622 - val_acc:
0.6421
Epoch 5/25
40000/40000 - 11s - loss: 1.0363 - acc: 0.6397 - val_loss: 0.9257 - val_acc:
0.6761
```

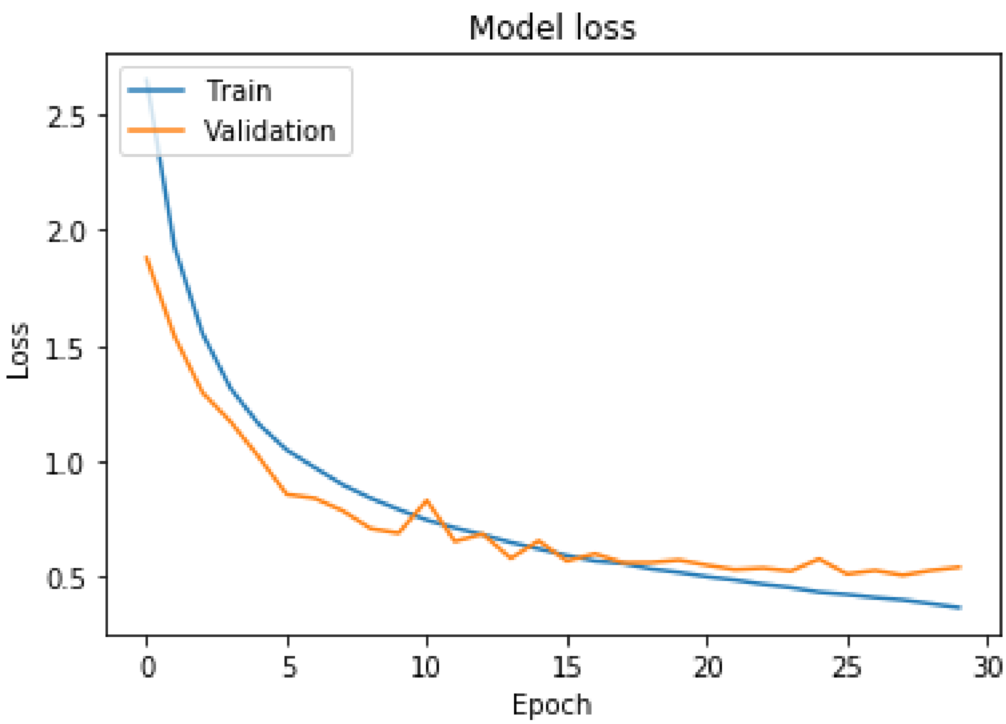
```
Epoch 6/25
40000/40000 - 11s - loss: 0.9441 - acc: 0.6712 - val_loss: 0.7996 - val_acc:
0.7215
Epoch 7/25
40000/40000 - 11s - loss: 0.8747 - acc: 0.6949 - val_loss: 0.7942 - val_acc:
0.7193
Epoch 8/25
40000/40000 - 11s - loss: 0.8148 - acc: 0.7177 - val_loss: 0.7697 - val_acc:
0.7230
Epoch 9/25
40000/40000 - 11s - loss: 0.7700 - acc: 0.7346 - val_loss: 0.7907 - val_acc:
0.7294
Epoch 10/25
40000/40000 - 11s - loss: 0.7267 - acc: 0.7487 - val_loss: 0.7276 - val_acc:
0.7484
Epoch 11/25
40000/40000 - 11s - loss: 0.6934 - acc: 0.7591 - val_loss: 0.6376 - val_acc:
0.7796
Epoch 12/25
40000/40000 - 11s - loss: 0.6618 - acc: 0.7711 - val_loss: 0.6953 - val_acc:
0.7604
Epoch 13/25
40000/40000 - 11s - loss: 0.6405 - acc: 0.7786 - val_loss: 0.6269 - val_acc:
0.7778
Epoch 14/25
40000/40000 - 11s - loss: 0.6096 - acc: 0.7897 - val_loss: 0.6706 - val_acc:
0.7675
Epoch 15/25
40000/40000 - 11s - loss: 0.5910 - acc: 0.7960 - val_loss: 0.5888 - val_acc:
0.7935
Epoch 16/25
40000/40000 - 11s - loss: 0.5651 - acc: 0.8027 - val_loss: 0.5636 - val_acc:
0.8073
Epoch 17/25
40000/40000 - 11s - loss: 0.5439 - acc: 0.8132 - val_loss: 0.5399 - val_acc:
0.8116
Epoch 18/25
40000/40000 - 11s - loss: 0.5281 - acc: 0.8179 - val_loss: 0.5330 - val_acc:
0.8156
Epoch 19/25
40000/40000 - 11s - loss: 0.5100 - acc: 0.8241 - val_loss: 0.5447 - val_acc:
0.8100
Epoch 20/25
40000/40000 - 11s - loss: 0.4936 - acc: 0.8315 - val_loss: 0.5456 - val_acc:
0.8158
Epoch 21/25
40000/40000 - 11s - loss: 0.4713 - acc: 0.8369 - val_loss: 0.4975 - val_acc:
0.8298
```

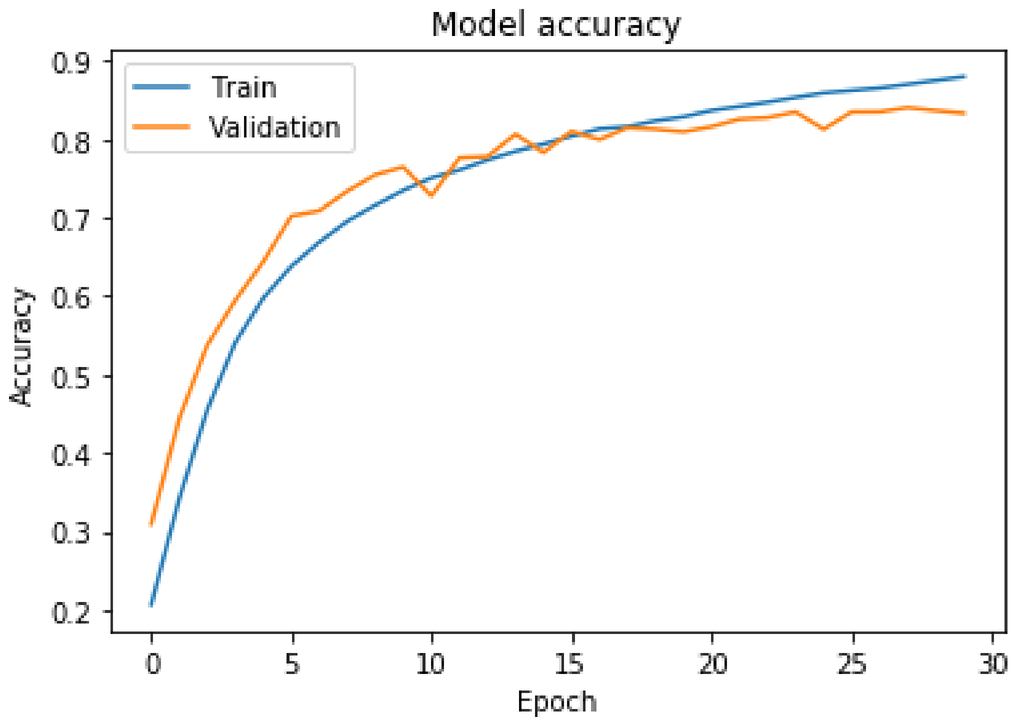
```
Epoch 22/25
40000/40000 - 11s - loss: 0.4656 - acc: 0.8394 - val_loss: 0.5683 - val_acc:
0.8054
Epoch 23/25
40000/40000 - 11s - loss: 0.4404 - acc: 0.8496 - val_loss: 0.5148 - val_acc:
0.8253
Epoch 24/25
40000/40000 - 11s - loss: 0.4306 - acc: 0.8505 - val_loss: 0.5101 - val_acc:
0.8321
Epoch 25/25
40000/40000 - 11s - loss: 0.4206 - acc: 0.8530 - val_loss: 0.5218 - val_acc:
0.8290
```

[59]:

```
print("Dropout increasing with normalization")
plot_graphs(history1)
```

```
Dropout increasing with normalization
```





We see that the model above is overfitting the training data. We will add more regularization (both dropout and weight decay).

Dropout increasing 0.15 ->0.45 and weight decay = 0.005 with normalization

[22]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2(),
                 input_shape=shape))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                  kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.15))

model.add(Conv2D(64, (3, 3),
                 padding='same',
```

```
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform',
                 kernel_regularizer=regularizers.l2()))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Dropout(0.45))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compile the model
opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history2 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
```

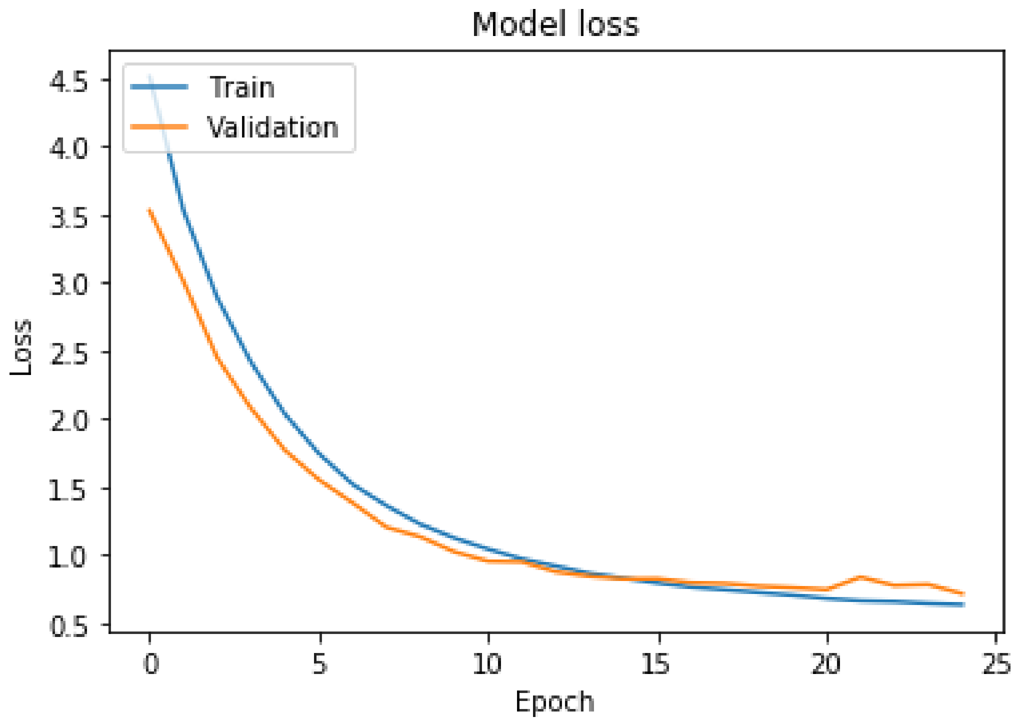
```
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_5_part2.h5', overwrite=True)
```

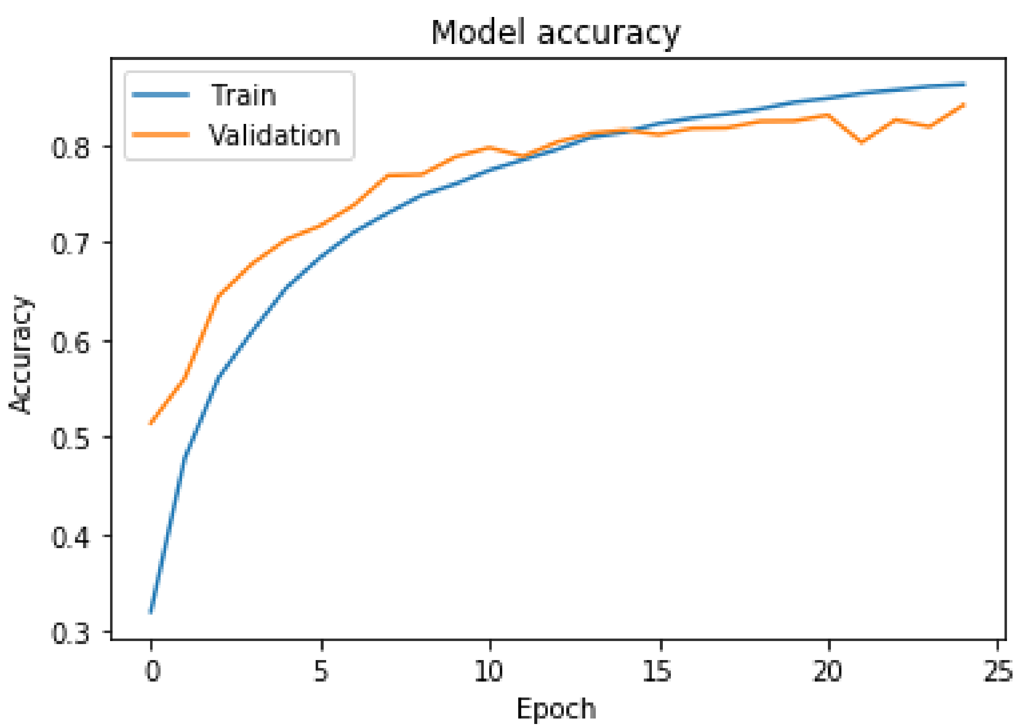
```
Train on 40000 samples, validate on 10000 samples
Epoch 1/25
40000/40000 - 13s - loss: 4.5092 - acc: 0.3197 - val_loss: 3.5261 - val_acc:
0.5139
Epoch 2/25
40000/40000 - 12s - loss: 3.5274 - acc: 0.4775 - val_loss: 3.0077 - val_acc:
0.5600
Epoch 3/25
40000/40000 - 12s - loss: 2.8864 - acc: 0.5605 - val_loss: 2.4447 - val_acc:
0.6447
Epoch 4/25
40000/40000 - 12s - loss: 2.4109 - acc: 0.6084 - val_loss: 2.0729 - val_acc:
0.6781
Epoch 5/25
40000/40000 - 12s - loss: 2.0308 - acc: 0.6532 - val_loss: 1.7648 - val_acc:
0.7029
Epoch 6/25
40000/40000 - 12s - loss: 1.7419 - acc: 0.6842 - val_loss: 1.5506 - val_acc:
0.7170
Epoch 7/25
40000/40000 - 12s - loss: 1.5166 - acc: 0.7105 - val_loss: 1.3802 - val_acc:
0.7383
Epoch 8/25
40000/40000 - 12s - loss: 1.3595 - acc: 0.7301 - val_loss: 1.2005 - val_acc:
0.7689
Epoch 9/25
40000/40000 - 12s - loss: 1.2229 - acc: 0.7485 - val_loss: 1.1307 - val_acc:
0.7699
Epoch 10/25
40000/40000 - 12s - loss: 1.1216 - acc: 0.7602 - val_loss: 1.0203 - val_acc:
0.7881
Epoch 11/25
40000/40000 - 12s - loss: 1.0398 - acc: 0.7743 - val_loss: 0.9515 - val_acc:
0.7976
Epoch 12/25
40000/40000 - 12s - loss: 0.9682 - acc: 0.7852 - val_loss: 0.9465 - val_acc:
0.7887
Epoch 13/25
40000/40000 - 12s - loss: 0.9151 - acc: 0.7957 - val_loss: 0.8735 - val_acc:
0.8033
Epoch 14/25
```

```
40000/40000 - 12s - loss: 0.8615 - acc: 0.8084 - val_loss: 0.8416 - val_acc:
0.8124
Epoch 15/25
40000/40000 - 12s - loss: 0.8267 - acc: 0.8134 - val_loss: 0.8219 - val_acc:
0.8157
Epoch 16/25
40000/40000 - 12s - loss: 0.7922 - acc: 0.8224 - val_loss: 0.8233 - val_acc:
0.8110
Epoch 17/25
40000/40000 - 12s - loss: 0.7622 - acc: 0.8281 - val_loss: 0.7941 - val_acc:
0.8175
Epoch 18/25
40000/40000 - 12s - loss: 0.7429 - acc: 0.8324 - val_loss: 0.7884 - val_acc:
0.8180
Epoch 19/25
40000/40000 - 12s - loss: 0.7221 - acc: 0.8374 - val_loss: 0.7691 - val_acc:
0.8247
Epoch 20/25
40000/40000 - 12s - loss: 0.7021 - acc: 0.8443 - val_loss: 0.7579 - val_acc:
0.8250
Epoch 21/25
40000/40000 - 12s - loss: 0.6784 - acc: 0.8485 - val_loss: 0.7434 - val_acc:
0.8312
Epoch 22/25
40000/40000 - 12s - loss: 0.6614 - acc: 0.8535 - val_loss: 0.8350 - val_acc:
0.8030
Epoch 23/25
40000/40000 - 12s - loss: 0.6556 - acc: 0.8569 - val_loss: 0.7741 - val_acc:
0.8262
Epoch 24/25
40000/40000 - 12s - loss: 0.6423 - acc: 0.8606 - val_loss: 0.7813 - val_acc:
0.8192
Epoch 25/25
40000/40000 - 12s - loss: 0.6329 - acc: 0.8626 - val_loss: 0.7140 - val_acc:
0.8414
```

[23]:
```
print("Dropout increasing from 0.15 -> 0.45, weight decay 0.0005, normalization␣
    ↪after each layer")
plot_graphs(history2)
```

```
Dropout increasing from 0.15 -> 0.45, weight decay 0.0005, normalization after
each layer
```


Model loss
Train
Validation
Loss
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
5
10
15
20
25
Epoch


Model accuracy
Train
Validation
Accuracy
0.8
0.7
0.6
0.5
0.4
0.3
0
5
10
15
20
25
Epoch

1.8.2 Validation

Looking at the graphs above as well as the graphs in our model #3 and model #4, normalization has improved both our accuracy and has reduced the validation/training loss. We will keep this in our model. We will also use the dropout increasing for the regularization because it reduces overfitting more than if we add weight decay.

1.9 Model #6 - Optimizers

In the first model, we arbitrarily chose the "RMSprop" optimizer. Now we will try several other optimizers to see which one produces the best results. However, we will also increase the number of epochs.

1.9.1 RMSprop

[0]:
```
epochs = 25
```

[27]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 input_shape=shape))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.15))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
```

```python
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Dropout(0.45))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compile the model
opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history1 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_6_part1.h5', overwrite=True)
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/50
40000/40000 - 13s - loss: 2.2848 - acc: 0.3016 - val_loss: 1.5005 - val_acc:
0.4640
Epoch 2/50
40000/40000 - 12s - loss: 1.6665 - acc: 0.4458 - val_loss: 1.3384 - val_acc:
0.5169
Epoch 3/50
40000/40000 - 12s - loss: 1.4028 - acc: 0.5221 - val_loss: 1.0763 - val_acc:
0.6161
Epoch 4/50
40000/40000 - 12s - loss: 1.2039 - acc: 0.5833 - val_loss: 1.0893 - val_acc:
0.6208
```

```
Epoch 5/50
40000/40000 - 12s - loss: 1.0739 - acc: 0.6280 - val_loss: 0.8896 - val_acc:
0.6819
Epoch 6/50
40000/40000 - 12s - loss: 0.9801 - acc: 0.6603 - val_loss: 0.8239 - val_acc:
0.7085
Epoch 7/50
40000/40000 - 12s - loss: 0.8989 - acc: 0.6848 - val_loss: 0.7973 - val_acc:
0.7216
Epoch 8/50
40000/40000 - 12s - loss: 0.8389 - acc: 0.7101 - val_loss: 0.7316 - val_acc:
0.7485
Epoch 9/50
40000/40000 - 12s - loss: 0.7892 - acc: 0.7254 - val_loss: 0.6843 - val_acc:
0.7588
Epoch 10/50
40000/40000 - 12s - loss: 0.7480 - acc: 0.7391 - val_loss: 0.7494 - val_acc:
0.7380
Epoch 11/50
40000/40000 - 12s - loss: 0.7120 - acc: 0.7510 - val_loss: 0.7443 - val_acc:
0.7381
Epoch 12/50
40000/40000 - 12s - loss: 0.6822 - acc: 0.7612 - val_loss: 0.6649 - val_acc:
0.7671
Epoch 13/50
40000/40000 - 11s - loss: 0.6493 - acc: 0.7737 - val_loss: 0.6046 - val_acc:
0.7851
Epoch 14/50
40000/40000 - 12s - loss: 0.6259 - acc: 0.7837 - val_loss: 0.5968 - val_acc:
0.7922
Epoch 15/50
40000/40000 - 11s - loss: 0.5990 - acc: 0.7928 - val_loss: 0.5811 - val_acc:
0.7962
Epoch 16/50
40000/40000 - 11s - loss: 0.5817 - acc: 0.7970 - val_loss: 0.5651 - val_acc:
0.8062
Epoch 17/50
40000/40000 - 12s - loss: 0.5573 - acc: 0.8062 - val_loss: 0.5762 - val_acc:
0.8045
Epoch 18/50
40000/40000 - 12s - loss: 0.5341 - acc: 0.8132 - val_loss: 0.5574 - val_acc:
0.8094
Epoch 19/50
40000/40000 - 12s - loss: 0.5200 - acc: 0.8192 - val_loss: 0.5734 - val_acc:
0.8012
Epoch 20/50
40000/40000 - 12s - loss: 0.5065 - acc: 0.8217 - val_loss: 0.5341 - val_acc:
0.8174
```

```
Epoch 21/50
40000/40000 - 12s - loss: 0.4886 - acc: 0.8320 - val_loss: 0.5383 - val_acc:
0.8165
Epoch 22/50
40000/40000 - 12s - loss: 0.4693 - acc: 0.8363 - val_loss: 0.5446 - val_acc:
0.8134
Epoch 23/50
40000/40000 - 12s - loss: 0.4551 - acc: 0.8422 - val_loss: 0.5428 - val_acc:
0.8164
Epoch 24/50
40000/40000 - 11s - loss: 0.4403 - acc: 0.8488 - val_loss: 0.5083 - val_acc:
0.8256
Epoch 25/50
40000/40000 - 12s - loss: 0.4250 - acc: 0.8525 - val_loss: 0.5178 - val_acc:
0.8239
Epoch 26/50
40000/40000 - 11s - loss: 0.4153 - acc: 0.8560 - val_loss: 0.5178 - val_acc:
0.8239
Epoch 27/50
40000/40000 - 12s - loss: 0.4023 - acc: 0.8578 - val_loss: 0.4998 - val_acc:
0.8312
Epoch 28/50
40000/40000 - 12s - loss: 0.3926 - acc: 0.8638 - val_loss: 0.5105 - val_acc:
0.8258
Epoch 29/50
40000/40000 - 11s - loss: 0.3823 - acc: 0.8662 - val_loss: 0.5172 - val_acc:
0.8263
Epoch 30/50
40000/40000 - 11s - loss: 0.3721 - acc: 0.8708 - val_loss: 0.5156 - val_acc:
0.8324
Epoch 31/50
40000/40000 - 12s - loss: 0.3631 - acc: 0.8730 - val_loss: 0.5158 - val_acc:
0.8291
Epoch 32/50
40000/40000 - 12s - loss: 0.3537 - acc: 0.8744 - val_loss: 0.5063 - val_acc:
0.8332
Epoch 33/50
40000/40000 - 12s - loss: 0.3429 - acc: 0.8781 - val_loss: 0.5079 - val_acc:
0.8354
Epoch 34/50
40000/40000 - 11s - loss: 0.3362 - acc: 0.8820 - val_loss: 0.5372 - val_acc:
0.8295
Epoch 35/50
40000/40000 - 11s - loss: 0.3273 - acc: 0.8868 - val_loss: 0.5313 - val_acc:
0.8329
Epoch 36/50
40000/40000 - 11s - loss: 0.3194 - acc: 0.8887 - val_loss: 0.5354 - val_acc:
0.8301
```

```
Epoch 37/50
40000/40000 - 11s - loss: 0.3119 - acc: 0.8922 - val_loss: 0.5313 - val_acc:
0.8284
Epoch 38/50
40000/40000 - 11s - loss: 0.3087 - acc: 0.8910 - val_loss: 0.5158 - val_acc:
0.8371
Epoch 39/50
40000/40000 - 11s - loss: 0.2958 - acc: 0.8959 - val_loss: 0.4947 - val_acc:
0.8403
Epoch 40/50
40000/40000 - 11s - loss: 0.2930 - acc: 0.8963 - val_loss: 0.5207 - val_acc:
0.8371
Epoch 41/50
40000/40000 - 11s - loss: 0.2902 - acc: 0.8985 - val_loss: 0.5481 - val_acc:
0.8279
Epoch 42/50
40000/40000 - 11s - loss: 0.2842 - acc: 0.9000 - val_loss: 0.5348 - val_acc:
0.8306
Epoch 43/50
40000/40000 - 11s - loss: 0.2786 - acc: 0.9022 - val_loss: 0.5022 - val_acc:
0.8409
Epoch 44/50
40000/40000 - 11s - loss: 0.2714 - acc: 0.9044 - val_loss: 0.5028 - val_acc:
0.8440
Epoch 45/50
40000/40000 - 11s - loss: 0.2627 - acc: 0.9070 - val_loss: 0.5343 - val_acc:
0.8343
Epoch 46/50
40000/40000 - 11s - loss: 0.2592 - acc: 0.9076 - val_loss: 0.5062 - val_acc:
0.8438
Epoch 47/50
40000/40000 - 11s - loss: 0.2552 - acc: 0.9105 - val_loss: 0.5159 - val_acc:
0.8442
Epoch 48/50
40000/40000 - 11s - loss: 0.2467 - acc: 0.9124 - val_loss: 0.5299 - val_acc:
0.8397
Epoch 49/50
40000/40000 - 11s - loss: 0.2438 - acc: 0.9139 - val_loss: 0.5036 - val_acc:
0.8468
Epoch 50/50
40000/40000 - 11s - loss: 0.2410 - acc: 0.9136 - val_loss: 0.5403 - val_acc:
0.8386
```

1.9.2 SGD - Stochastic gradient descent optimizer

[32]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 input_shape=shape))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.15))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Dropout(0.45))
```

```python
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compile the model
opt = keras.optimizers.SGD(learning_rate=0.01, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history2 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_6_part2.h5', overwrite=True)
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/50
40000/40000 - 14s - loss: 1.8051 - acc: 0.3739 - val_loss: 1.4150 - val_acc:
0.5021
Epoch 2/50
40000/40000 - 11s - loss: 1.3103 - acc: 0.5346 - val_loss: 1.0651 - val_acc:
0.6146
Epoch 3/50
40000/40000 - 11s - loss: 1.0742 - acc: 0.6229 - val_loss: 0.9073 - val_acc:
0.6793
Epoch 4/50
40000/40000 - 11s - loss: 0.9253 - acc: 0.6745 - val_loss: 0.7608 - val_acc:
0.7305
Epoch 5/50
40000/40000 - 11s - loss: 0.8235 - acc: 0.7096 - val_loss: 0.6986 - val_acc:
0.7562
Epoch 6/50
40000/40000 - 11s - loss: 0.7481 - acc: 0.7397 - val_loss: 0.6321 - val_acc:
0.7782
Epoch 7/50
40000/40000 - 11s - loss: 0.6873 - acc: 0.7604 - val_loss: 0.6395 - val_acc:
0.7756
Epoch 8/50
40000/40000 - 11s - loss: 0.6429 - acc: 0.7775 - val_loss: 0.5805 - val_acc:
0.8006
Epoch 9/50
40000/40000 - 11s - loss: 0.6040 - acc: 0.7907 - val_loss: 0.5761 - val_acc:
```

```
0.8011
Epoch 10/50
40000/40000 - 11s - loss: 0.5760 - acc: 0.8007 - val_loss: 0.5828 - val_acc:
0.7995
Epoch 11/50
40000/40000 - 11s - loss: 0.5420 - acc: 0.8122 - val_loss: 0.5641 - val_acc:
0.8053
Epoch 12/50
40000/40000 - 11s - loss: 0.5152 - acc: 0.8198 - val_loss: 0.5771 - val_acc:
0.8036
Epoch 13/50
40000/40000 - 11s - loss: 0.4873 - acc: 0.8316 - val_loss: 0.5179 - val_acc:
0.8218
Epoch 14/50
40000/40000 - 11s - loss: 0.4650 - acc: 0.8379 - val_loss: 0.5403 - val_acc:
0.8156
Epoch 15/50
40000/40000 - 11s - loss: 0.4489 - acc: 0.8433 - val_loss: 0.5159 - val_acc:
0.8278
Epoch 16/50
40000/40000 - 11s - loss: 0.4346 - acc: 0.8487 - val_loss: 0.5162 - val_acc:
0.8297
Epoch 17/50
40000/40000 - 11s - loss: 0.4115 - acc: 0.8559 - val_loss: 0.4904 - val_acc:
0.8379
Epoch 18/50
40000/40000 - 11s - loss: 0.4002 - acc: 0.8603 - val_loss: 0.5236 - val_acc:
0.8275
Epoch 19/50
40000/40000 - 11s - loss: 0.3815 - acc: 0.8654 - val_loss: 0.4962 - val_acc:
0.8407
Epoch 20/50
40000/40000 - 11s - loss: 0.3722 - acc: 0.8704 - val_loss: 0.4971 - val_acc:
0.8368
Epoch 21/50
40000/40000 - 11s - loss: 0.3562 - acc: 0.8753 - val_loss: 0.5031 - val_acc:
0.8358
Epoch 22/50
40000/40000 - 11s - loss: 0.3445 - acc: 0.8787 - val_loss: 0.5070 - val_acc:
0.8395
Epoch 23/50
40000/40000 - 11s - loss: 0.3322 - acc: 0.8841 - val_loss: 0.4973 - val_acc:
0.8416
Epoch 24/50
40000/40000 - 11s - loss: 0.3246 - acc: 0.8857 - val_loss: 0.5115 - val_acc:
0.8385
Epoch 25/50
40000/40000 - 11s - loss: 0.3080 - acc: 0.8924 - val_loss: 0.4983 - val_acc:
```

```
0.8450
Epoch 26/50
40000/40000 - 11s - loss: 0.3045 - acc: 0.8927 - val_loss: 0.4913 - val_acc:
0.8465
Epoch 27/50
40000/40000 - 11s - loss: 0.2944 - acc: 0.8961 - val_loss: 0.5057 - val_acc:
0.8419
Epoch 28/50
40000/40000 - 11s - loss: 0.2881 - acc: 0.8982 - val_loss: 0.5035 - val_acc:
0.8393
Epoch 29/50
40000/40000 - 11s - loss: 0.2751 - acc: 0.9019 - val_loss: 0.5169 - val_acc:
0.8406
Epoch 30/50
40000/40000 - 11s - loss: 0.2797 - acc: 0.9007 - val_loss: 0.5031 - val_acc:
0.8456
Epoch 31/50
40000/40000 - 11s - loss: 0.2511 - acc: 0.9115 - val_loss: 0.5202 - val_acc:
0.8424
Epoch 32/50
40000/40000 - 11s - loss: 0.2589 - acc: 0.9068 - val_loss: 0.5153 - val_acc:
0.8435
Epoch 33/50
40000/40000 - 11s - loss: 0.2536 - acc: 0.9103 - val_loss: 0.4926 - val_acc:
0.8462
Epoch 34/50
40000/40000 - 11s - loss: 0.2432 - acc: 0.9157 - val_loss: 0.5137 - val_acc:
0.8429
Epoch 35/50
40000/40000 - 11s - loss: 0.2414 - acc: 0.9144 - val_loss: 0.5184 - val_acc:
0.8430
Epoch 36/50
40000/40000 - 11s - loss: 0.2319 - acc: 0.9173 - val_loss: 0.5304 - val_acc:
0.8451
Epoch 37/50
40000/40000 - 11s - loss: 0.2241 - acc: 0.9207 - val_loss: 0.5480 - val_acc:
0.8421
Epoch 38/50
40000/40000 - 11s - loss: 0.2213 - acc: 0.9213 - val_loss: 0.5259 - val_acc:
0.8492
Epoch 39/50
40000/40000 - 11s - loss: 0.2194 - acc: 0.9230 - val_loss: 0.5425 - val_acc:
0.8415
Epoch 40/50
40000/40000 - 11s - loss: 0.2139 - acc: 0.9245 - val_loss: 0.5644 - val_acc:
0.8389
Epoch 41/50
40000/40000 - 11s - loss: 0.2119 - acc: 0.9262 - val_loss: 0.5224 - val_acc:
```

```
0.8475
Epoch 42/50
40000/40000 - 11s - loss: 0.2064 - acc: 0.9263 - val_loss: 0.5293 - val_acc:
0.8477
Epoch 43/50
40000/40000 - 11s - loss: 0.1942 - acc: 0.9309 - val_loss: 0.5854 - val_acc:
0.8330
Epoch 44/50
40000/40000 - 11s - loss: 0.1969 - acc: 0.9313 - val_loss: 0.5293 - val_acc:
0.8499
Epoch 45/50
40000/40000 - 11s - loss: 0.1918 - acc: 0.9326 - val_loss: 0.5236 - val_acc:
0.8505
Epoch 46/50
40000/40000 - 11s - loss: 0.1869 - acc: 0.9331 - val_loss: 0.5280 - val_acc:
0.8508
Epoch 47/50
40000/40000 - 11s - loss: 0.1831 - acc: 0.9360 - val_loss: 0.5356 - val_acc:
0.8486
Epoch 48/50
40000/40000 - 11s - loss: 0.1847 - acc: 0.9352 - val_loss: 0.5336 - val_acc:
0.8486
Epoch 49/50
40000/40000 - 11s - loss: 0.1772 - acc: 0.9367 - val_loss: 0.5371 - val_acc:
0.8508
Epoch 50/50
40000/40000 - 11s - loss: 0.1842 - acc: 0.9358 - val_loss: 0.5303 - val_acc:
0.8479
```

1.9.3 Adagrad

[33]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 input_shape=shape))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.15))

model.add(Conv2D(64, (3, 3),
                 padding='same',
```

```
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Dropout(0.45))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compile the model
opt = keras.optimizers.Adagrad(learning_rate=0.01)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history3 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
```

```
model.save('model_6_part3.h5', overwrite=True)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow_core/python/keras/optimizer_v2/adagrad.py:107: calling
Constant.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated
and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor
Train on 40000 samples, validate on 10000 samples
Epoch 1/50
40000/40000 - 14s - loss: 1.7443 - acc: 0.4086 - val_loss: 1.1958 - val_acc:
0.5661
Epoch 2/50
40000/40000 - 11s - loss: 1.2376 - acc: 0.5635 - val_loss: 1.0010 - val_acc:
0.6428
Epoch 3/50
40000/40000 - 11s - loss: 1.0434 - acc: 0.6277 - val_loss: 0.9030 - val_acc:
0.6806
Epoch 4/50
40000/40000 - 11s - loss: 0.9333 - acc: 0.6696 - val_loss: 0.8655 - val_acc:
0.6870
Epoch 5/50
40000/40000 - 11s - loss: 0.8611 - acc: 0.6963 - val_loss: 0.7300 - val_acc:
0.7430
Epoch 6/50
40000/40000 - 11s - loss: 0.7979 - acc: 0.7198 - val_loss: 0.7097 - val_acc:
0.7463
Epoch 7/50
40000/40000 - 11s - loss: 0.7545 - acc: 0.7372 - val_loss: 0.6592 - val_acc:
0.7662
Epoch 8/50
40000/40000 - 11s - loss: 0.7194 - acc: 0.7486 - val_loss: 0.6989 - val_acc:
0.7541
Epoch 9/50
40000/40000 - 11s - loss: 0.6848 - acc: 0.7596 - val_loss: 0.6479 - val_acc:
0.7731
Epoch 10/50
40000/40000 - 11s - loss: 0.6593 - acc: 0.7691 - val_loss: 0.6236 - val_acc:
0.7756
Epoch 11/50
40000/40000 - 11s - loss: 0.6315 - acc: 0.7789 - val_loss: 0.5948 - val_acc:
0.7866
Epoch 12/50
40000/40000 - 11s - loss: 0.6144 - acc: 0.7870 - val_loss: 0.5938 - val_acc:
0.7936
Epoch 13/50
```

```
40000/40000 - 11s - loss: 0.5892 - acc: 0.7926 - val_loss: 0.5849 - val_acc:
0.7922
Epoch 14/50
40000/40000 - 11s - loss: 0.5647 - acc: 0.8026 - val_loss: 0.5564 - val_acc:
0.8021
Epoch 15/50
40000/40000 - 11s - loss: 0.5514 - acc: 0.8062 - val_loss: 0.5543 - val_acc:
0.8079
Epoch 16/50
40000/40000 - 11s - loss: 0.5351 - acc: 0.8125 - val_loss: 0.5460 - val_acc:
0.8093
Epoch 17/50
40000/40000 - 11s - loss: 0.5205 - acc: 0.8192 - val_loss: 0.5562 - val_acc:
0.8082
Epoch 18/50
40000/40000 - 11s - loss: 0.5066 - acc: 0.8255 - val_loss: 0.5189 - val_acc:
0.8208
Epoch 19/50
40000/40000 - 11s - loss: 0.4895 - acc: 0.8285 - val_loss: 0.5255 - val_acc:
0.8189
Epoch 20/50
40000/40000 - 11s - loss: 0.4760 - acc: 0.8337 - val_loss: 0.5282 - val_acc:
0.8207
Epoch 21/50
40000/40000 - 11s - loss: 0.4620 - acc: 0.8392 - val_loss: 0.5274 - val_acc:
0.8187
Epoch 22/50
40000/40000 - 11s - loss: 0.4535 - acc: 0.8402 - val_loss: 0.5203 - val_acc:
0.8223
Epoch 23/50
40000/40000 - 11s - loss: 0.4370 - acc: 0.8471 - val_loss: 0.5064 - val_acc:
0.8267
Epoch 24/50
40000/40000 - 11s - loss: 0.4344 - acc: 0.8474 - val_loss: 0.5283 - val_acc:
0.8212
Epoch 25/50
40000/40000 - 11s - loss: 0.4255 - acc: 0.8499 - val_loss: 0.5124 - val_acc:
0.8254
Epoch 26/50
40000/40000 - 11s - loss: 0.4102 - acc: 0.8574 - val_loss: 0.5051 - val_acc:
0.8284
Epoch 27/50
40000/40000 - 11s - loss: 0.4009 - acc: 0.8582 - val_loss: 0.5119 - val_acc:
0.8291
Epoch 28/50
40000/40000 - 11s - loss: 0.3886 - acc: 0.8637 - val_loss: 0.5042 - val_acc:
0.8296
Epoch 29/50
```

```
40000/40000 - 11s - loss: 0.3837 - acc: 0.8654 - val_loss: 0.4996 - val_acc:
0.8344
Epoch 30/50
40000/40000 - 11s - loss: 0.3725 - acc: 0.8672 - val_loss: 0.5034 - val_acc:
0.8320
Epoch 31/50
40000/40000 - 11s - loss: 0.3674 - acc: 0.8701 - val_loss: 0.4858 - val_acc:
0.8377
Epoch 32/50
40000/40000 - 11s - loss: 0.3519 - acc: 0.8765 - val_loss: 0.4984 - val_acc:
0.8354
Epoch 33/50
40000/40000 - 11s - loss: 0.3530 - acc: 0.8757 - val_loss: 0.5067 - val_acc:
0.8316
Epoch 34/50
40000/40000 - 11s - loss: 0.3425 - acc: 0.8793 - val_loss: 0.4968 - val_acc:
0.8351
Epoch 35/50
40000/40000 - 11s - loss: 0.3347 - acc: 0.8813 - val_loss: 0.5172 - val_acc:
0.8325
Epoch 36/50
40000/40000 - 11s - loss: 0.3324 - acc: 0.8838 - val_loss: 0.4867 - val_acc:
0.8433
Epoch 37/50
40000/40000 - 11s - loss: 0.3244 - acc: 0.8859 - val_loss: 0.4922 - val_acc:
0.8407
Epoch 38/50
40000/40000 - 11s - loss: 0.3117 - acc: 0.8906 - val_loss: 0.4969 - val_acc:
0.8421
Epoch 39/50
40000/40000 - 11s - loss: 0.3096 - acc: 0.8890 - val_loss: 0.4969 - val_acc:
0.8432
Epoch 40/50
40000/40000 - 11s - loss: 0.3002 - acc: 0.8946 - val_loss: 0.4989 - val_acc:
0.8398
Epoch 41/50
40000/40000 - 11s - loss: 0.2980 - acc: 0.8958 - val_loss: 0.4880 - val_acc:
0.8432
Epoch 42/50
40000/40000 - 11s - loss: 0.2921 - acc: 0.8976 - val_loss: 0.4988 - val_acc:
0.8422
Epoch 43/50
40000/40000 - 11s - loss: 0.2865 - acc: 0.9004 - val_loss: 0.5134 - val_acc:
0.8356
Epoch 44/50
40000/40000 - 11s - loss: 0.2809 - acc: 0.9018 - val_loss: 0.4947 - val_acc:
0.8444
Epoch 45/50
```

```
40000/40000 - 11s - loss: 0.2729 - acc: 0.9044 - val_loss: 0.5020 - val_acc:
0.8424
Epoch 46/50
40000/40000 - 11s - loss: 0.2689 - acc: 0.9068 - val_loss: 0.5023 - val_acc:
0.8399
Epoch 47/50
40000/40000 - 11s - loss: 0.2634 - acc: 0.9075 - val_loss: 0.4968 - val_acc:
0.8431
Epoch 48/50
40000/40000 - 11s - loss: 0.2592 - acc: 0.9096 - val_loss: 0.4944 - val_acc:
0.8433
Epoch 49/50
40000/40000 - 11s - loss: 0.2557 - acc: 0.9089 - val_loss: 0.5066 - val_acc:
0.8426
Epoch 50/50
40000/40000 - 11s - loss: 0.2571 - acc: 0.9094 - val_loss: 0.5079 - val_acc:
0.8434
```

1.9.4 Adadelta

A robust extension of Adagrad that adapts learning rates based on a moving window of gradient updates (source https://keras.io/optimizers/)

[34]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 input_shape=shape))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.15))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Dropout(0.45))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compile the model
opt = keras.optimizers.Adadelta(learning_rate=1.0, rho=0.95)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history4 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_6_part4.h5', overwrite=True)
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/50
40000/40000 - 14s - loss: 1.6527 - acc: 0.4318 - val_loss: 1.2876 - val_acc:
0.5391
Epoch 2/50
40000/40000 - 11s - loss: 1.0363 - acc: 0.6357 - val_loss: 1.2510 - val_acc:
0.6011
```

```
Epoch 3/50
40000/40000 - 11s - loss: 0.8458 - acc: 0.7068 - val_loss: 0.7598 - val_acc:
0.7344
Epoch 4/50
40000/40000 - 11s - loss: 0.7398 - acc: 0.7442 - val_loss: 0.6771 - val_acc:
0.7661
Epoch 5/50
40000/40000 - 11s - loss: 0.6627 - acc: 0.7688 - val_loss: 0.7105 - val_acc:
0.7532
Epoch 6/50
40000/40000 - 11s - loss: 0.6178 - acc: 0.7850 - val_loss: 0.6000 - val_acc:
0.7932
Epoch 7/50
40000/40000 - 11s - loss: 0.5658 - acc: 0.8049 - val_loss: 0.5570 - val_acc:
0.8109
Epoch 8/50
40000/40000 - 11s - loss: 0.5270 - acc: 0.8176 - val_loss: 0.5395 - val_acc:
0.8187
Epoch 9/50
40000/40000 - 11s - loss: 0.4941 - acc: 0.8307 - val_loss: 0.5526 - val_acc:
0.8123
Epoch 10/50
40000/40000 - 11s - loss: 0.4726 - acc: 0.8379 - val_loss: 0.5618 - val_acc:
0.8110
Epoch 11/50
40000/40000 - 11s - loss: 0.4443 - acc: 0.8475 - val_loss: 0.5709 - val_acc:
0.8114
Epoch 12/50
40000/40000 - 11s - loss: 0.4203 - acc: 0.8557 - val_loss: 0.5107 - val_acc:
0.8330
Epoch 13/50
40000/40000 - 11s - loss: 0.4035 - acc: 0.8623 - val_loss: 0.5200 - val_acc:
0.8287
Epoch 14/50
40000/40000 - 11s - loss: 0.3859 - acc: 0.8670 - val_loss: 0.5502 - val_acc:
0.8262
Epoch 15/50
40000/40000 - 11s - loss: 0.3711 - acc: 0.8714 - val_loss: 0.5321 - val_acc:
0.8315
Epoch 16/50
40000/40000 - 11s - loss: 0.3503 - acc: 0.8789 - val_loss: 0.4910 - val_acc:
0.8417
Epoch 17/50
40000/40000 - 11s - loss: 0.3402 - acc: 0.8831 - val_loss: 0.5171 - val_acc:
0.8364
Epoch 18/50
40000/40000 - 11s - loss: 0.3265 - acc: 0.8860 - val_loss: 0.5020 - val_acc:
0.8373
```

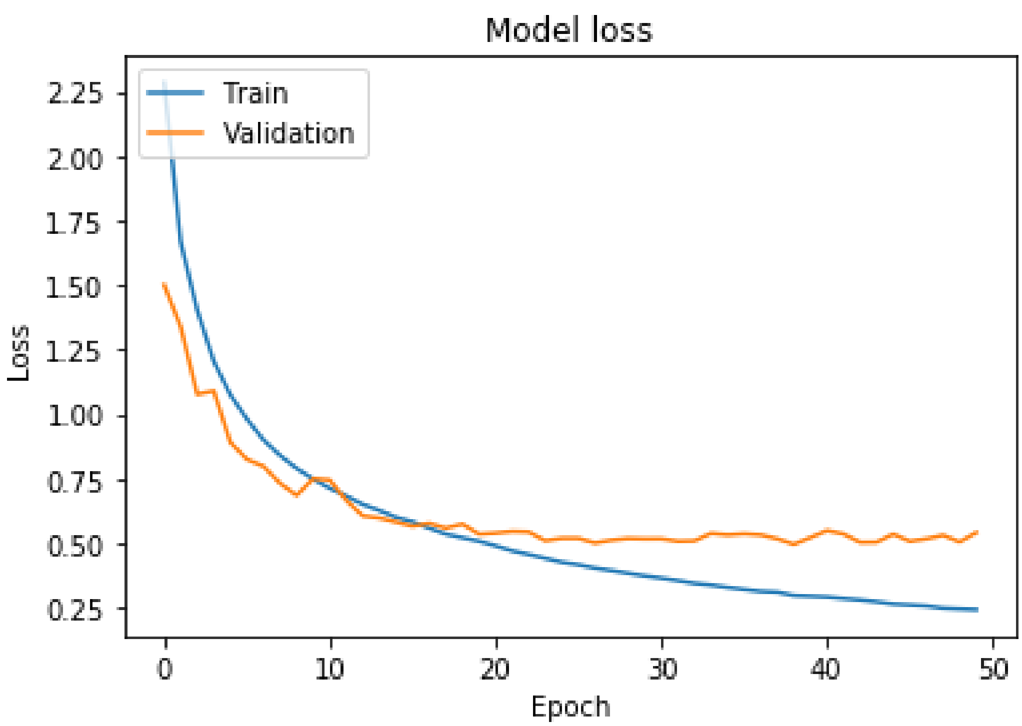
```
Epoch 19/50
40000/40000 - 11s - loss: 0.3121 - acc: 0.8915 - val_loss: 0.5363 - val_acc:
0.8391
Epoch 20/50
40000/40000 - 11s - loss: 0.2993 - acc: 0.8956 - val_loss: 0.5646 - val_acc:
0.8231
Epoch 21/50
40000/40000 - 11s - loss: 0.2847 - acc: 0.9020 - val_loss: 0.4825 - val_acc:
0.8510
Epoch 22/50
40000/40000 - 11s - loss: 0.2776 - acc: 0.9036 - val_loss: 0.5395 - val_acc:
0.8403
Epoch 23/50
40000/40000 - 11s - loss: 0.2743 - acc: 0.9063 - val_loss: 0.4968 - val_acc:
0.8508
Epoch 24/50
40000/40000 - 11s - loss: 0.2637 - acc: 0.9086 - val_loss: 0.4972 - val_acc:
0.8458
Epoch 25/50
40000/40000 - 12s - loss: 0.2583 - acc: 0.9098 - val_loss: 0.5082 - val_acc:
0.8468
Epoch 26/50
40000/40000 - 11s - loss: 0.2534 - acc: 0.9117 - val_loss: 0.5320 - val_acc:
0.8427
Epoch 27/50
40000/40000 - 11s - loss: 0.2458 - acc: 0.9143 - val_loss: 0.5404 - val_acc:
0.8434
Epoch 28/50
40000/40000 - 11s - loss: 0.2371 - acc: 0.9156 - val_loss: 0.5814 - val_acc:
0.8276
Epoch 29/50
40000/40000 - 11s - loss: 0.2320 - acc: 0.9208 - val_loss: 0.5013 - val_acc:
0.8544
Epoch 30/50
40000/40000 - 11s - loss: 0.2275 - acc: 0.9227 - val_loss: 0.5433 - val_acc:
0.8436
Epoch 31/50
40000/40000 - 11s - loss: 0.2180 - acc: 0.9232 - val_loss: 0.5044 - val_acc:
0.8472
Epoch 32/50
40000/40000 - 11s - loss: 0.2220 - acc: 0.9236 - val_loss: 0.5363 - val_acc:
0.8463
Epoch 33/50
40000/40000 - 11s - loss: 0.2143 - acc: 0.9259 - val_loss: 0.5084 - val_acc:
0.8565
Epoch 34/50
40000/40000 - 11s - loss: 0.2097 - acc: 0.9272 - val_loss: 0.5568 - val_acc:
0.8501
```

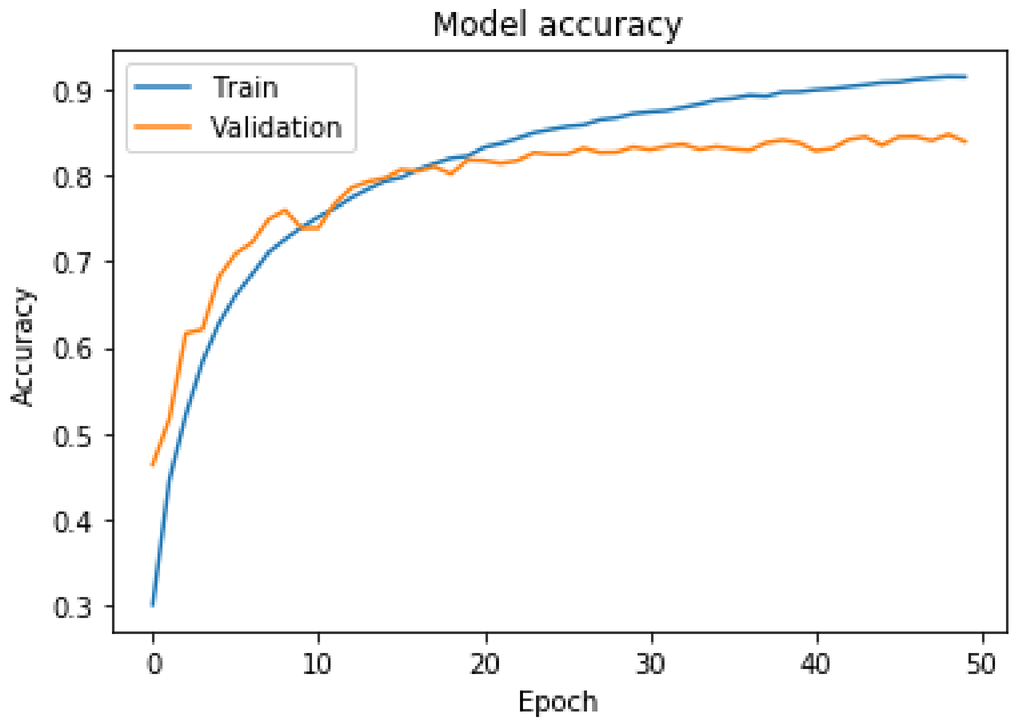
```
Epoch 35/50
40000/40000 - 11s - loss: 0.2025 - acc: 0.9285 - val_loss: 0.5613 - val_acc:
0.8474
Epoch 36/50
40000/40000 - 11s - loss: 0.1968 - acc: 0.9324 - val_loss: 0.5926 - val_acc:
0.8372
Epoch 37/50
40000/40000 - 11s - loss: 0.2007 - acc: 0.9306 - val_loss: 0.5479 - val_acc:
0.8485
Epoch 38/50
40000/40000 - 11s - loss: 0.1877 - acc: 0.9364 - val_loss: 0.5358 - val_acc:
0.8560
Epoch 39/50
40000/40000 - 11s - loss: 0.1918 - acc: 0.9350 - val_loss: 0.5357 - val_acc:
0.8490
Epoch 40/50
40000/40000 - 11s - loss: 0.1891 - acc: 0.9354 - val_loss: 0.5832 - val_acc:
0.8414
Epoch 41/50
40000/40000 - 11s - loss: 0.1877 - acc: 0.9356 - val_loss: 0.5358 - val_acc:
0.8520
Epoch 42/50
40000/40000 - 11s - loss: 0.1756 - acc: 0.9390 - val_loss: 0.5488 - val_acc:
0.8492
Epoch 43/50
40000/40000 - 11s - loss: 0.1803 - acc: 0.9386 - val_loss: 0.5241 - val_acc:
0.8595
Epoch 44/50
40000/40000 - 11s - loss: 0.1733 - acc: 0.9403 - val_loss: 0.5071 - val_acc:
0.8530
Epoch 45/50
40000/40000 - 11s - loss: 0.1651 - acc: 0.9423 - val_loss: 0.5281 - val_acc:
0.8617
Epoch 46/50
40000/40000 - 11s - loss: 0.1699 - acc: 0.9407 - val_loss: 0.5472 - val_acc:
0.8531
Epoch 47/50
40000/40000 - 11s - loss: 0.1614 - acc: 0.9442 - val_loss: 0.5655 - val_acc:
0.8527
Epoch 48/50
40000/40000 - 11s - loss: 0.1637 - acc: 0.9434 - val_loss: 0.5456 - val_acc:
0.8527
Epoch 49/50
40000/40000 - 11s - loss: 0.1667 - acc: 0.9421 - val_loss: 0.5326 - val_acc:
0.8531
Epoch 50/50
40000/40000 - 11s - loss: 0.1609 - acc: 0.9449 - val_loss: 0.5690 - val_acc:
0.8490
```

1.9.5 Validation

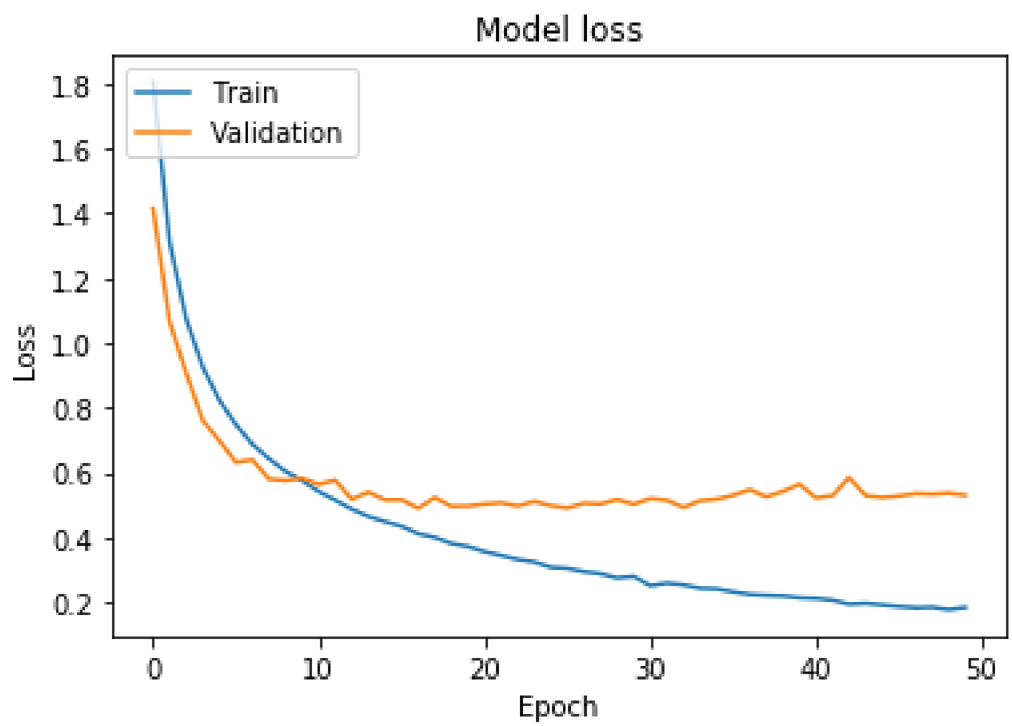
[36]:
```
print("Optimizers/n")

print("RMSprop")
plot_graphs(history1)

print("SGD")
plot_graphs(history2)

print("Adagrad")
plot_graphs(history3)

print("Adadelta")
plot_graphs(history4)
```

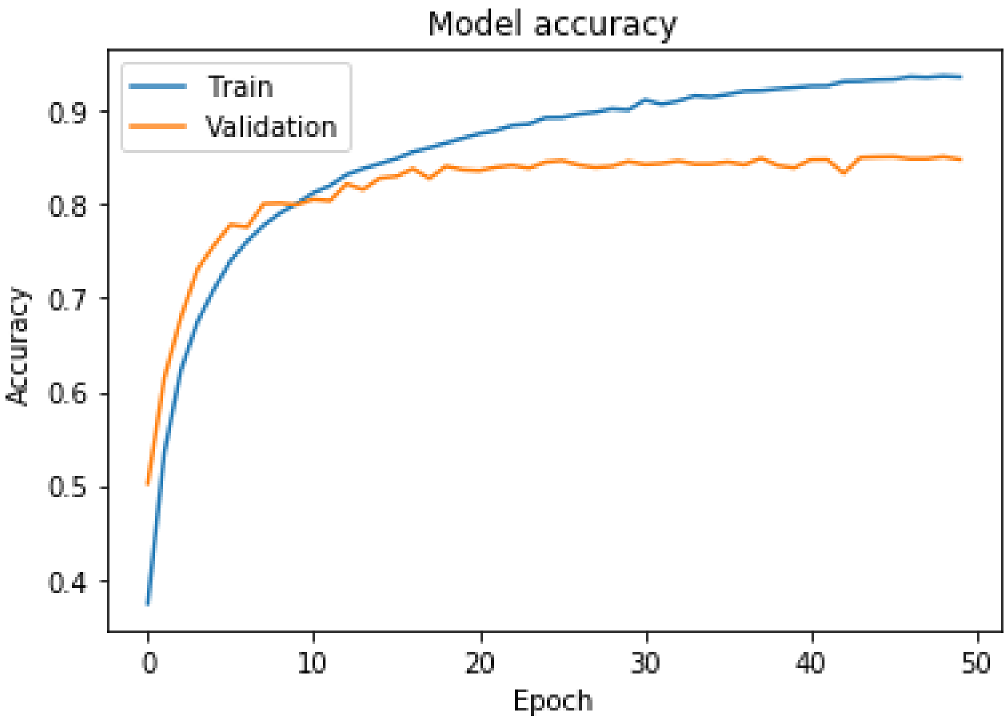
```
Optimizers/n
RMSprop
```



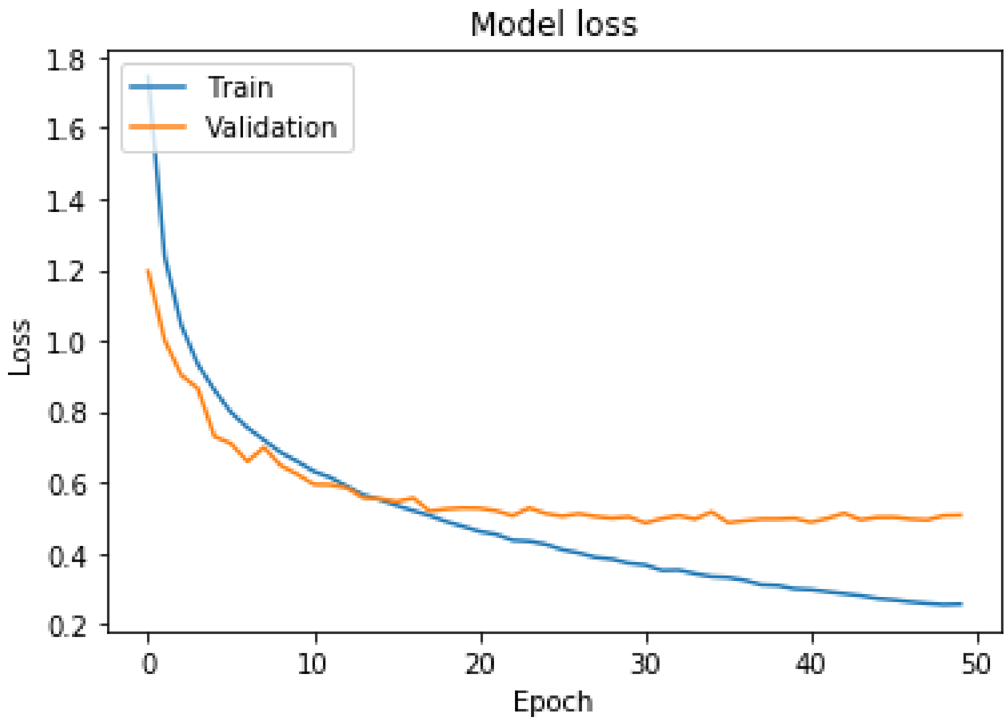

Model accuracy
Train
Validation
Accuracy
Epoch

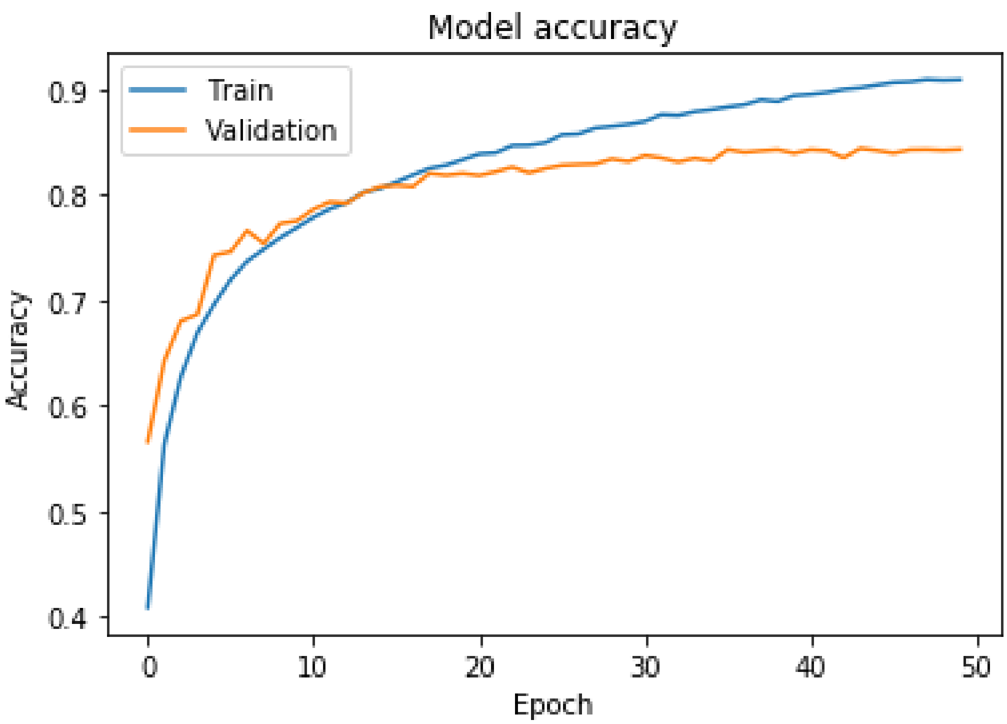
SGD


Model loss
Train
Validation
Loss
Epoch

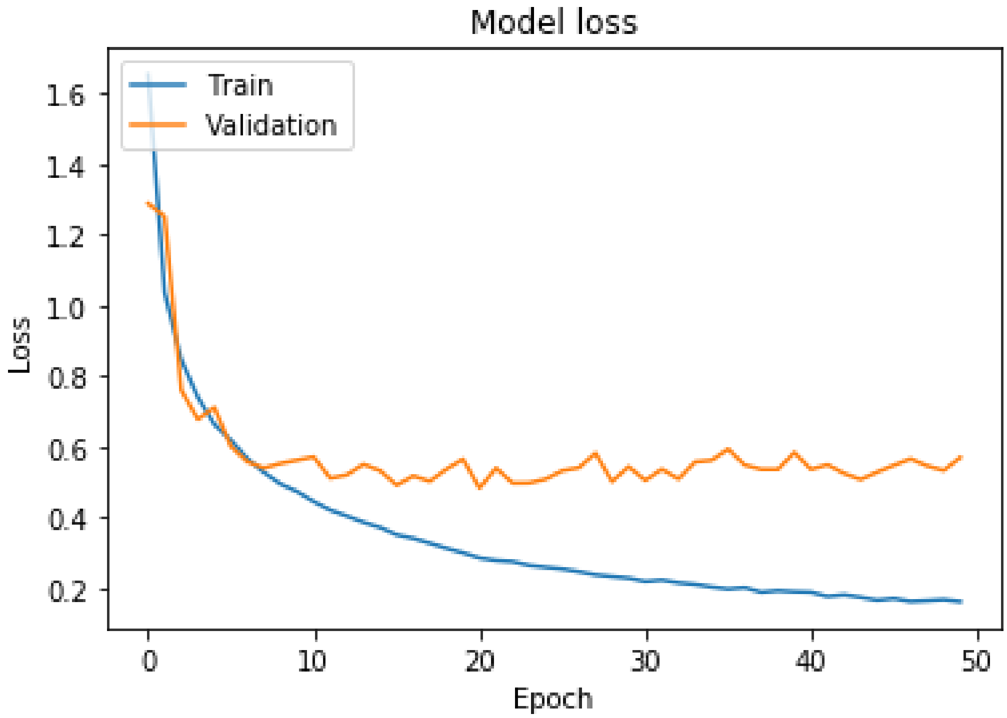

Model accuracy
Train
Validation
0.9
0.8
0.7
0.6
0.5
0.4
Accuracy
0
10
20
30
40
50
Epoch

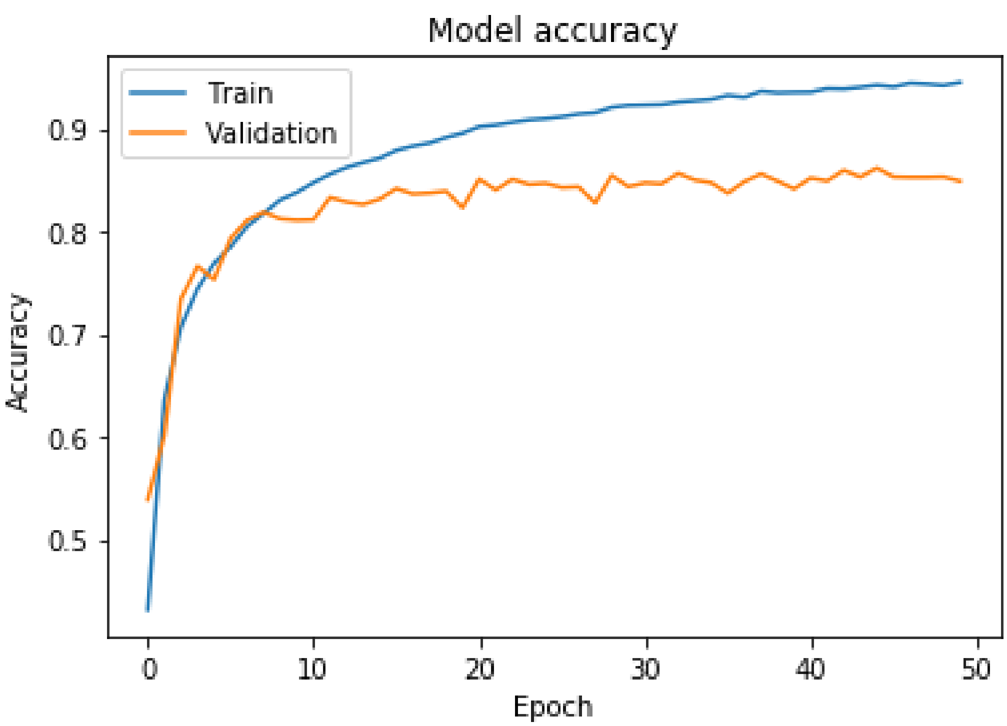
Adagrad


Model loss
Train
Validation
Loss
Epoch
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
10
20
30
40
50


Model accuracy
Train
Validation
Accuracy
Epoch
0.9
0.8
0.7
0.6
0.5
0.4
0
10
20
30
40
50

Adadelta


Model loss
Train
Validation
Loss
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
10
20
30
40
50
Epoch


Model accuracy
Train
Validation
Accuracy
0.9
0.8
0.7
0.6
0.5
0
10
20
30
40
50
Epoch

1.9.6 Validation Results

From the graphs as well as the output per model, we can see that all optimizers tend to start overfitting after around 25 epochs. To counteract this, we need to increase the regularization (dropout layer) or we can limit the epochs to 25. Since the accuracy doesn't seem to increase much after 25, we will keep it around 25.

At around epoch 25, all the optimizers seem to perform very similarly, with small differences with increase in accuracy for slightly more overfitted models. To create our final models, we will stick to 30 for number of epochs and RMSprop and Adadelta for our optimizers (best fit model and most accurate model) and slightly increase the regularization.

1.10 Model #7 - Final touches

For our final models, we will increase the number of layers to see if that increases our accuracy and reduces the training/validation loss. Along with the increased number of layers, based on model #6, we will increase the number of epochs slightly to 30 along with slightly higher regularization (dropout increases from 0.25 -> 0.55)

[0]:
```
epochs = 30
```

[70]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 input_shape=shape))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35))

model.add(Conv2D(256, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(256, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.45))

model.add(Conv2D(512, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(512, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.55))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Dropout(0.55))
model.add(Dense(num_classes))
```

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

# Compile the model
opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

#Train model
history1 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_7_part1.h5', overwrite=True)
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/30
40000/40000 - 36s - loss: 2.8870 - acc: 0.1779 - val_loss: 1.9347 - val_acc:
0.2806
Epoch 2/30
40000/40000 - 24s - loss: 2.0701 - acc: 0.3055 - val_loss: 1.5718 - val_acc:
0.4171
Epoch 3/30
40000/40000 - 25s - loss: 1.6754 - acc: 0.4223 - val_loss: 1.3655 - val_acc:
0.5190
Epoch 4/30
40000/40000 - 25s - loss: 1.4175 - acc: 0.5077 - val_loss: 1.3039 - val_acc:
0.5410
Epoch 5/30
40000/40000 - 25s - loss: 1.2404 - acc: 0.5747 - val_loss: 1.0291 - val_acc:
0.6402
Epoch 6/30
40000/40000 - 25s - loss: 1.1190 - acc: 0.6182 - val_loss: 0.9412 - val_acc:
0.6748
Epoch 7/30
40000/40000 - 25s - loss: 1.0191 - acc: 0.6506 - val_loss: 0.8903 - val_acc:
0.6860
Epoch 8/30
40000/40000 - 25s - loss: 0.9466 - acc: 0.6790 - val_loss: 0.7647 - val_acc:
0.7351
Epoch 9/30
40000/40000 - 24s - loss: 0.8887 - acc: 0.7010 - val_loss: 0.7762 - val_acc:
0.7299
```

```
Epoch 10/30
40000/40000 - 25s - loss: 0.8385 - acc: 0.7187 - val_loss: 0.7193 - val_acc:
0.7529
Epoch 11/30
40000/40000 - 25s - loss: 0.7910 - acc: 0.7353 - val_loss: 0.7048 - val_acc:
0.7597
Epoch 12/30
40000/40000 - 25s - loss: 0.7547 - acc: 0.7483 - val_loss: 0.6544 - val_acc:
0.7776
Epoch 13/30
40000/40000 - 25s - loss: 0.7214 - acc: 0.7585 - val_loss: 0.7536 - val_acc:
0.7537
Epoch 14/30
40000/40000 - 24s - loss: 0.6889 - acc: 0.7692 - val_loss: 0.6857 - val_acc:
0.7720
Epoch 15/30
40000/40000 - 24s - loss: 0.6523 - acc: 0.7836 - val_loss: 0.5984 - val_acc:
0.7982
Epoch 16/30
40000/40000 - 25s - loss: 0.6256 - acc: 0.7937 - val_loss: 0.5703 - val_acc:
0.8098
Epoch 17/30
40000/40000 - 25s - loss: 0.6172 - acc: 0.7968 - val_loss: 0.6295 - val_acc:
0.7896
Epoch 18/30
40000/40000 - 25s - loss: 0.5907 - acc: 0.8069 - val_loss: 0.5685 - val_acc:
0.8105
Epoch 19/30
40000/40000 - 25s - loss: 0.5732 - acc: 0.8111 - val_loss: 0.5549 - val_acc:
0.8194
Epoch 20/30
40000/40000 - 25s - loss: 0.5493 - acc: 0.8195 - val_loss: 0.5348 - val_acc:
0.8194
Epoch 21/30
40000/40000 - 25s - loss: 0.5338 - acc: 0.8266 - val_loss: 0.5462 - val_acc:
0.8202
Epoch 22/30
40000/40000 - 25s - loss: 0.5148 - acc: 0.8315 - val_loss: 0.5565 - val_acc:
0.8171
Epoch 23/30
40000/40000 - 25s - loss: 0.4995 - acc: 0.8362 - val_loss: 0.5320 - val_acc:
0.8241
Epoch 24/30
40000/40000 - 25s - loss: 0.4877 - acc: 0.8418 - val_loss: 0.5564 - val_acc:
0.8207
Epoch 25/30
40000/40000 - 25s - loss: 0.4763 - acc: 0.8466 - val_loss: 0.4911 - val_acc:
0.8417
```

```
Epoch 26/30
40000/40000 - 25s - loss: 0.4530 - acc: 0.8534 - val_loss: 0.6040 - val_acc:
0.8073
Epoch 27/30
40000/40000 - 25s - loss: 0.4447 - acc: 0.8556 - val_loss: 0.5570 - val_acc:
0.8210
Epoch 28/30
40000/40000 - 25s - loss: 0.4236 - acc: 0.8624 - val_loss: 0.5740 - val_acc:
0.8232
Epoch 29/30
40000/40000 - 24s - loss: 0.4209 - acc: 0.8632 - val_loss: 0.5200 - val_acc:
0.8390
Epoch 30/30
40000/40000 - 25s - loss: 0.4073 - acc: 0.8682 - val_loss: 0.4936 - val_acc:
0.8437
```

[73]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 kernel_initializer='random_uniform',
                 input_shape=shape))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.15))

model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
```

```
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.35))

model.add(Conv2D(256, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(256, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.45))

model.add(Conv2D(512, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(512, (3, 3),
                 padding='same',
                 kernel_initializer='random_uniform'))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.55))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Dropout(0.55))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# Compile the model
opt = keras.optimizers.Adadelta(learning_rate=1.0, rho=0.95)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
```

```python
             metrics=['accuracy'])

#Train model
history2 = model.fit(x_train,
          to_categorical(y_train, num_classes),
          epochs=epochs,
          verbose=2,
          validation_split=0.2,
          shuffle=True)

#Save model
model.save('model_7_part2.h5', overwrite=True)
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/30
40000/40000 - 36s - loss: 2.0485 - acc: 0.3242 - val_loss: 1.6751 - val_acc:
0.4436
Epoch 2/30
40000/40000 - 23s - loss: 1.2756 - acc: 0.5503 - val_loss: 1.0582 - val_acc:
0.6302
Epoch 3/30
40000/40000 - 23s - loss: 1.0445 - acc: 0.6426 - val_loss: 0.9104 - val_acc:
0.6892
Epoch 4/30
40000/40000 - 23s - loss: 0.8943 - acc: 0.6961 - val_loss: 0.7240 - val_acc:
0.7513
Epoch 5/30
40000/40000 - 23s - loss: 0.7978 - acc: 0.7339 - val_loss: 0.6624 - val_acc:
0.7795
Epoch 6/30
40000/40000 - 23s - loss: 0.7175 - acc: 0.7616 - val_loss: 0.6401 - val_acc:
0.7817
Epoch 7/30
40000/40000 - 23s - loss: 0.6604 - acc: 0.7804 - val_loss: 0.5858 - val_acc:
0.8040
Epoch 8/30
40000/40000 - 23s - loss: 0.6089 - acc: 0.7995 - val_loss: 0.6338 - val_acc:
0.7917
Epoch 9/30
40000/40000 - 23s - loss: 0.5638 - acc: 0.8143 - val_loss: 0.5560 - val_acc:
0.8157
Epoch 10/30
40000/40000 - 23s - loss: 0.5292 - acc: 0.8255 - val_loss: 0.5361 - val_acc:
0.8158
Epoch 11/30
40000/40000 - 23s - loss: 0.4995 - acc: 0.8352 - val_loss: 0.5433 - val_acc:
0.8184
```

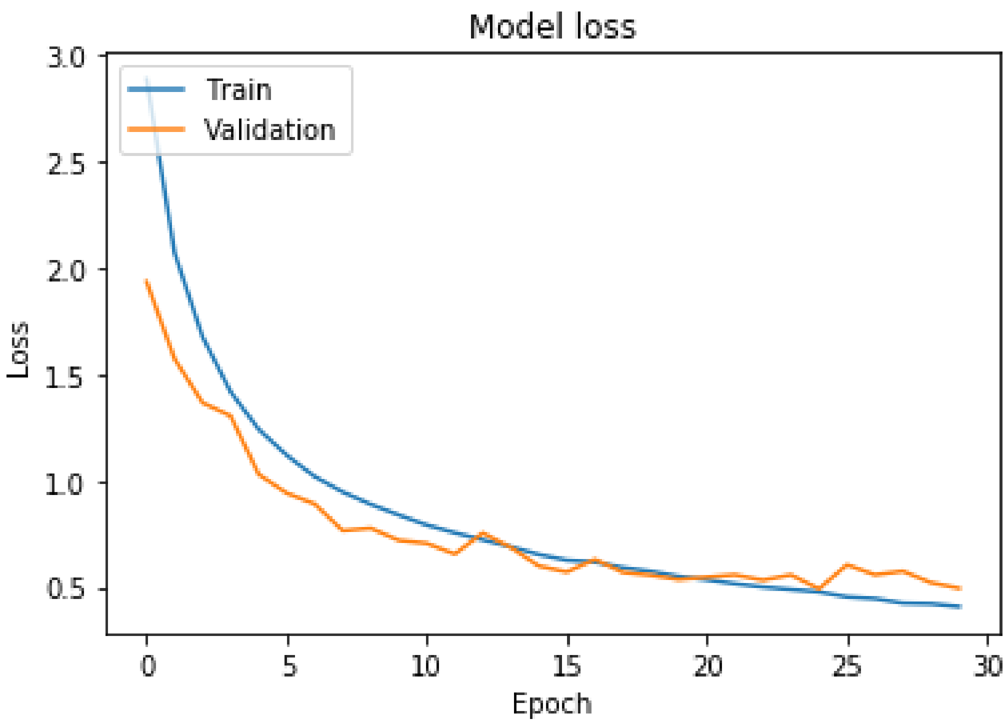
```
Epoch 12/30
40000/40000 - 23s - loss: 0.4661 - acc: 0.8478 - val_loss: 0.5173 - val_acc:
0.8312
Epoch 13/30
40000/40000 - 23s - loss: 0.4432 - acc: 0.8544 - val_loss: 0.5869 - val_acc:
0.8107
Epoch 14/30
40000/40000 - 23s - loss: 0.4177 - acc: 0.8634 - val_loss: 0.5087 - val_acc:
0.8316
Epoch 15/30
40000/40000 - 23s - loss: 0.3956 - acc: 0.8687 - val_loss: 0.6058 - val_acc:
0.8141
Epoch 16/30
40000/40000 - 23s - loss: 0.3772 - acc: 0.8745 - val_loss: 0.5051 - val_acc:
0.8367
Epoch 17/30
40000/40000 - 23s - loss: 0.3570 - acc: 0.8813 - val_loss: 0.4805 - val_acc:
0.8471
Epoch 18/30
40000/40000 - 23s - loss: 0.3386 - acc: 0.8882 - val_loss: 0.5333 - val_acc:
0.8385
Epoch 19/30
40000/40000 - 23s - loss: 0.3305 - acc: 0.8903 - val_loss: 0.4965 - val_acc:
0.8454
Epoch 20/30
40000/40000 - 22s - loss: 0.3110 - acc: 0.8972 - val_loss: 0.4838 - val_acc:
0.8525
Epoch 21/30
40000/40000 - 23s - loss: 0.2988 - acc: 0.9022 - val_loss: 0.5181 - val_acc:
0.8424
Epoch 22/30
40000/40000 - 23s - loss: 0.2868 - acc: 0.9075 - val_loss: 0.5428 - val_acc:
0.8396
Epoch 23/30
40000/40000 - 23s - loss: 0.2795 - acc: 0.9075 - val_loss: 0.5098 - val_acc:
0.8466
Epoch 24/30
40000/40000 - 23s - loss: 0.2619 - acc: 0.9128 - val_loss: 0.5514 - val_acc:
0.8351
Epoch 25/30
40000/40000 - 23s - loss: 0.2532 - acc: 0.9188 - val_loss: 0.5214 - val_acc:
0.8516
Epoch 26/30
40000/40000 - 23s - loss: 0.2503 - acc: 0.9183 - val_loss: 0.4986 - val_acc:
0.8493
Epoch 27/30
40000/40000 - 22s - loss: 0.2406 - acc: 0.9194 - val_loss: 0.5347 - val_acc:
0.8470
```

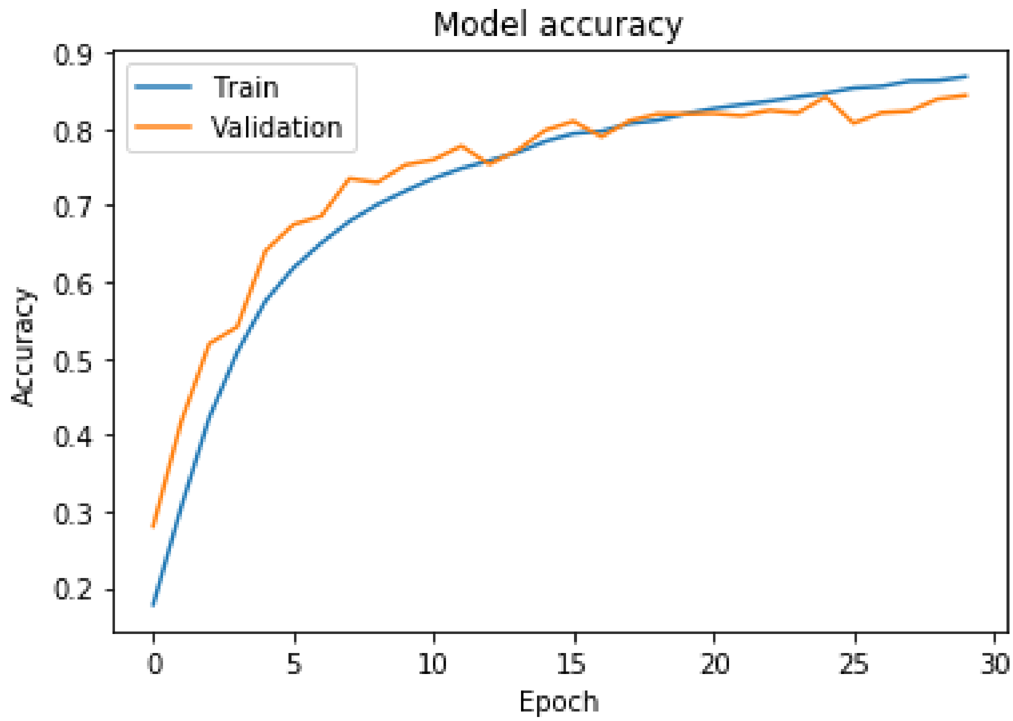
```
Epoch 28/30
40000/40000 - 22s - loss: 0.2307 - acc: 0.9241 - val_loss: 0.5404 - val_acc:
0.8401
Epoch 29/30
40000/40000 - 23s - loss: 0.2256 - acc: 0.9261 - val_loss: 0.5513 - val_acc:
0.8466
Epoch 30/30
40000/40000 - 23s - loss: 0.2129 - acc: 0.9307 - val_loss: 0.5880 - val_acc:
0.8449
```

1.10.1 Validation

[74]:
```
print("RMSprop")
plot_graphs(history1)

print("Adadelta")
plot_graphs(history2)
```

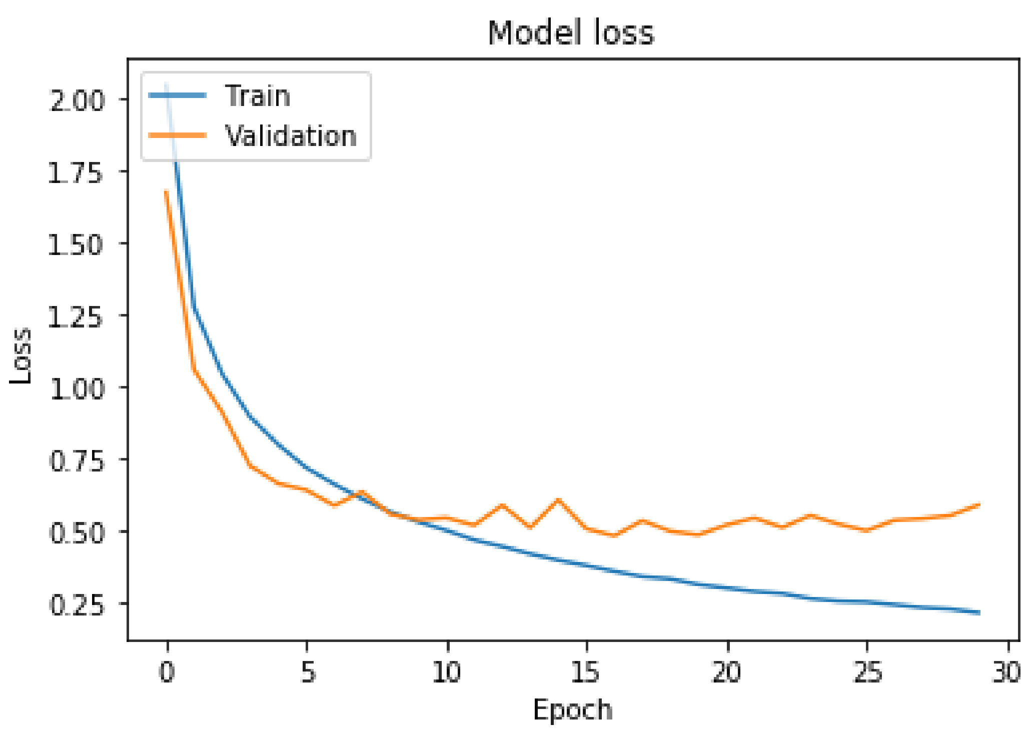
```
RMSprop
```



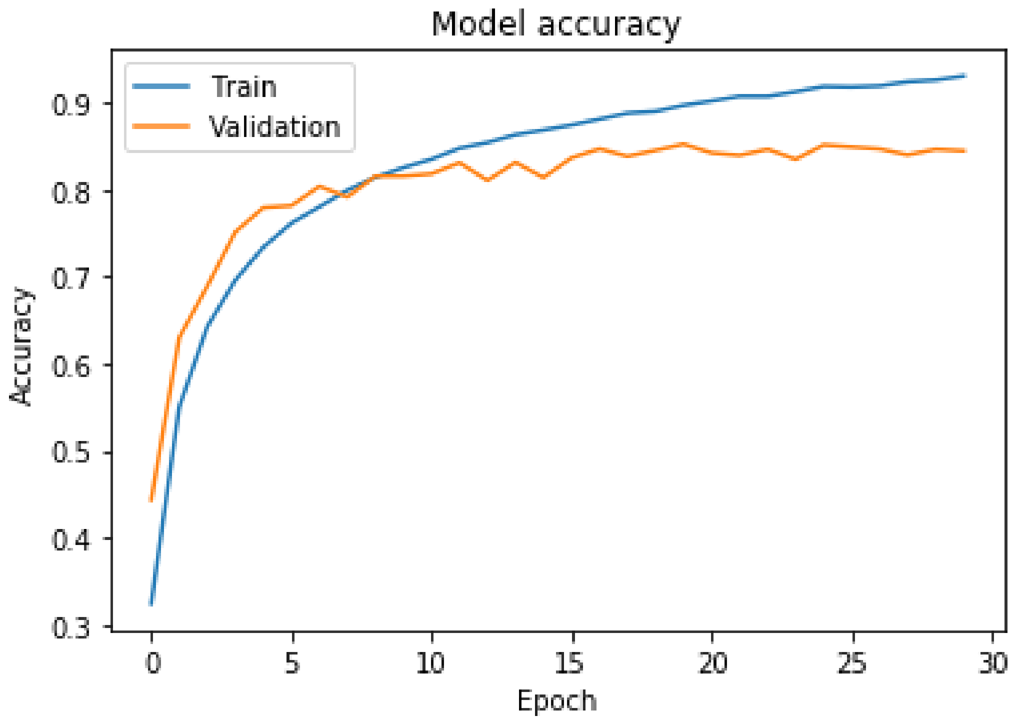


Model accuracy
Train
Validation
Accuracy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
5
10
15
20
25
30
Epoch

Adadelta



Model loss
Train
Validation
Loss
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0
5
10
15
20
25
30
Epoch

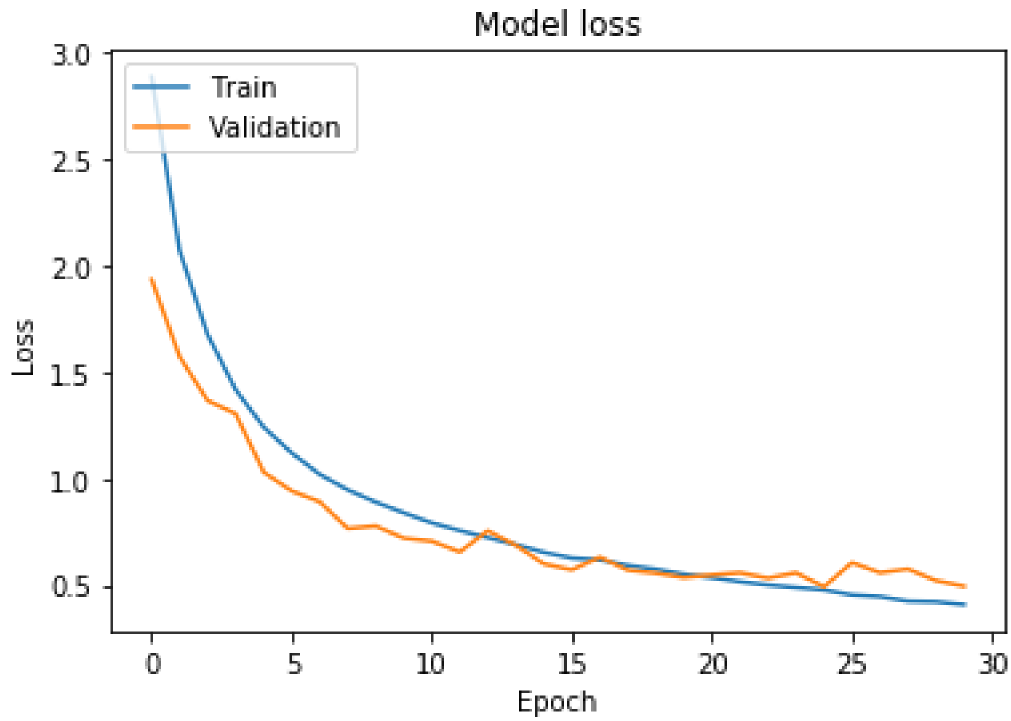


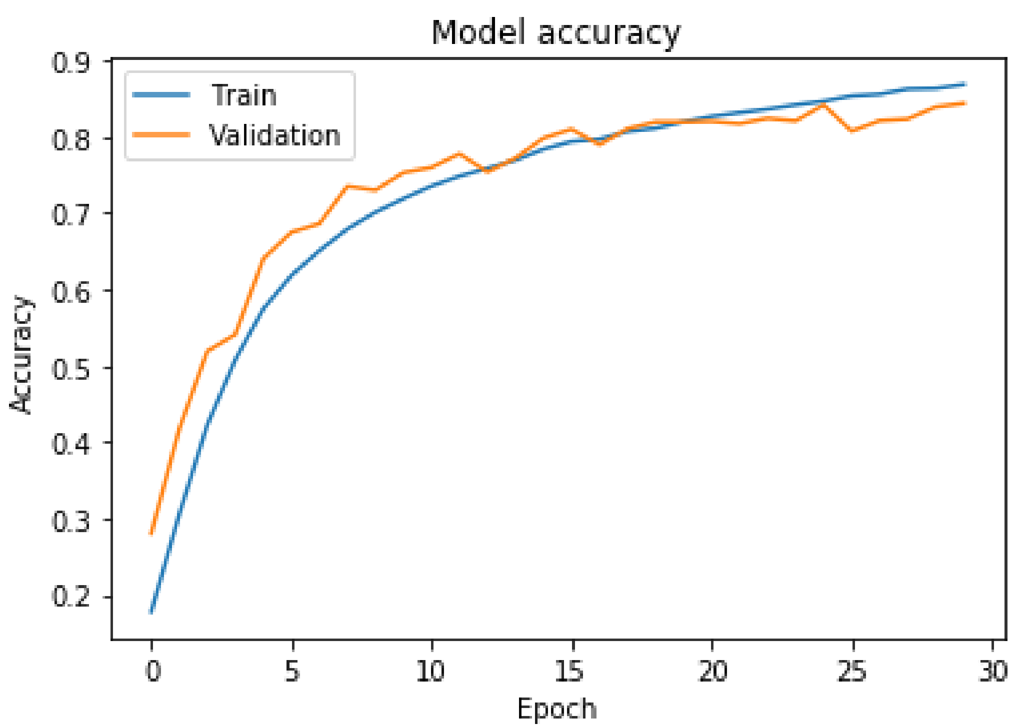
1.11 Final Model (Choosing the best one)

The final model we will chooose is model #7 part 1. This model uses the optimizer RMSprop, uses an increasing dropout of 0.25 -> 0.55 regularizer, Batch Normalization after every layer, 30 epochs with batch size 32, and 10 cnn layers followed by a flattening layer.

Net architecture is printed below along with training/validation accuracy and loss graph.

[77]:
```
plot_graphs(history1)
model = load_model("model_7_part1.h5")
print(model.summary())
```


Model loss
Train
Validation
Loss
Epoch
3.0
2.5
2.0
1.5
1.0
0.5
0
5
10
15
20
25
30


Model accuracy
Train
Validation
Accuracy
Epoch
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
5
10
15
20
25
30

```
Model: "sequential_29"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_204 (Conv2D)          (None, 32, 32, 32)        896
_________________________________________________________________
activation_254 (Activation)  (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_220 (Bat (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_205 (Conv2D)          (None, 32, 32, 32)        9248
_________________________________________________________________
activation_255 (Activation)  (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_221 (Bat (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_101 (MaxPoolin (None, 16, 16, 32)        0
_________________________________________________________________
dropout_122 (Dropout)        (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_206 (Conv2D)          (None, 16, 16, 64)        18496
_________________________________________________________________
activation_256 (Activation)  (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_222 (Bat (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_207 (Conv2D)          (None, 16, 16, 64)        36928
_________________________________________________________________
activation_257 (Activation)  (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_223 (Bat (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_102 (MaxPoolin (None, 8, 8, 64)          0
_________________________________________________________________
dropout_123 (Dropout)        (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_208 (Conv2D)          (None, 8, 8, 128)         73856
_________________________________________________________________
activation_258 (Activation)  (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_224 (Bat (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_209 (Conv2D)          (None, 8, 8, 128)         147584
_________________________________________________________________
activation_259 (Activation)  (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_225 (Bat (None, 8, 8, 128)         512
_________________________________________________________________
```

```
max_pooling2d_103 (MaxPoolin (None, 4, 4, 128)         0
_________________________________________________________________
dropout_124 (Dropout)        (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_210 (Conv2D)          (None, 4, 4, 256)         295168
_________________________________________________________________
activation_260 (Activation)  (None, 4, 4, 256)         0
_________________________________________________________________
batch_normalization_226 (Bat (None, 4, 4, 256)         1024
_________________________________________________________________
conv2d_211 (Conv2D)          (None, 4, 4, 256)         590080
_________________________________________________________________
activation_261 (Activation)  (None, 4, 4, 256)         0
_________________________________________________________________
batch_normalization_227 (Bat (None, 4, 4, 256)         1024
_________________________________________________________________
max_pooling2d_104 (MaxPoolin (None, 2, 2, 256)         0
_________________________________________________________________
dropout_125 (Dropout)        (None, 2, 2, 256)         0
_________________________________________________________________
conv2d_212 (Conv2D)          (None, 2, 2, 512)         1180160
_________________________________________________________________
activation_262 (Activation)  (None, 2, 2, 512)         0
_________________________________________________________________
batch_normalization_228 (Bat (None, 2, 2, 512)         2048
_________________________________________________________________
conv2d_213 (Conv2D)          (None, 2, 2, 512)         2359808
_________________________________________________________________
activation_263 (Activation)  (None, 2, 2, 512)         0
_________________________________________________________________
batch_normalization_229 (Bat (None, 2, 2, 512)         2048
_________________________________________________________________
max_pooling2d_105 (MaxPoolin (None, 1, 1, 512)         0
_________________________________________________________________
dropout_126 (Dropout)        (None, 1, 1, 512)         0
_________________________________________________________________
flatten_25 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_50 (Dense)             (None, 128)               65664
_________________________________________________________________
activation_264 (Activation)  (None, 128)               0
_________________________________________________________________
batch_normalization_230 (Bat (None, 128)               512
_________________________________________________________________
dropout_127 (Dropout)        (None, 128)               0
_________________________________________________________________
dense_51 (Dense)             (None, 10)                1290
_________________________________________________________________
```

```
activation_265 (Activation)  (None, 10)              0
=================================================================
Total params: 4,787,626
Trainable params: 4,783,402
Non-trainable params: 4,224
_________________________________________________________________
None
```

1.12 Testing Against Final Model

We will finally test our best model using the test data that we saved in the beginning. The following is the classification report and confusion matrix (created using helper functions in the second defined in the second section - useful helper functions).

[80]:
```python
get_metrics("model_7_part1.h5")
```

```
10000/10000 [==============================] - 6s 626us/sample - loss: 0.5163 -
acc: 0.8375
[0.5162984648227692, 0.8375]
              precision    recall  f1-score   support

    airplane       0.83      0.86      0.84      1000
  automobile       0.92      0.93      0.92      1000
        bird       0.82      0.72      0.77      1000
         cat       0.71      0.64      0.68      1000
        deer       0.82      0.85      0.84      1000
         dog       0.75      0.77      0.76      1000
        frog       0.84      0.90      0.87      1000
       horse       0.88      0.88      0.88      1000
        ship       0.91      0.89      0.90      1000
       truck       0.87      0.93      0.90      1000

    accuracy                           0.84     10000
   macro avg       0.84      0.84      0.84     10000
weighted avg       0.84      0.84      0.84     10000

            predicted - airplane  predicted - automobile  predicted - bird  \
airplane                     862                      13                16
automobile                     3                     926                 3
bird                          65                       0               723
cat                           22                       1                41
deer                          10                       1                34
dog                            9                       3                27
frog                           5                       2                18
horse                         11                       2                16
ship                          44                      22                 2
truck                         10                      36                 1
```

	predicted - cat	predicted - deer	predicted - dog
airplane	8	6	2
automobile	1	2	1
bird	39	54	44
cat	643	39	150
deer	30	851	14
dog	113	30	765
frog	32	19	11
horse	25	28	29
ship	8	3	0
truck	6	0	0

	predicted - frog	predicted - horse	predicted - ship
airplane	9	6	42
automobile	3	0	6
bird	55	10	5
cat	61	26	11
deer	23	33	4
dog	13	35	1
frog	902	7	4
horse	2	880	1
ship	3	0	892
truck	4	3	9

	predicted - truck
airplane	36
automobile	55
bird	5
cat	6
deer	0
dog	4
frog	0
horse	6
ship	26
truck	931

2 Conclusion/Results

In this project, we started by processing our data set and dividing our data into our training set and our test set. With our training set, we divided this into 80% training and 20% validation for each model (using the validation_split=0.2 variable when training the training data).

In our first model, we created a 6 layer cnn along with a flatten layer. We used an arbitrary optimizer (RMSprop) to begin with. As seen by the training/validation loss graph, our model was highly overfitting the data. Thus, regularization was needed.

With our second, third, and fourth models, we tried various parameters to regularize our data. In our second model, we tried using different weight decays to regularize (0, 0.00005, 0.0005, and 0.05). In our third model, we tried using different dropout layers (0.25, 0.5, and an increasing 0.15->0.45). In our fourth model, we used a combination of weight decay and dropout. With all these regularization techniques, it was determined that either the increasing dropout layer or the increasing dropout layer plus weight decay of 0.0005 best prevented overfitting while keeping accuracy up.

In our fifth model, we added normalization using Batch Normalization. Batch Normalization was added to each previous hidden layer at each batch and worked to reduce some overfitting and improve accuracy overall.

In our sixth model, we tried using different optimizers and epoch size 50. In the end, epochs of size 50 started overfitting the data and the good mixture of accuracy and preventing overfitting seemed to be around 30 epochs. The optimizers seemed to be relatively similar in performance, with RMSprop having the least difference between the training and validation loss and AdaDelta having the best overall accuracy on the validation data.

In our seventh and final model sets, we tried adjusting the epoch number to 35 as well as increasing the 6 layer cnn to 10 layers (plus the flatten layer).

Overall, the seventh model's first model seemed to have the best mixture of accuracy and minimizing the training and validation model loss. Thus, we chose this as our final model and ran this against our test set that we saved in the beginning.

2.1 Results

As stated in the Final Model section, the final model we chose and ran our test against is model #7 part 1. This model uses the optimizer RMSprop, uses an increasing dropout of 0.25 -> 0.55 for regularization, Batch Normalization after every layer, 30 epochs with batch size 32, and 10 cnn layers followed by a flattening layer.

The net architecture is as following:

[81]:
```
model = load_model("model_7_part1.h5")
print(model.summary())
```

```
Model: "sequential_29"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_204 (Conv2D)          (None, 32, 32, 32)        896
_________________________________________________________________
activation_254 (Activation)  (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_220 (Bat (None, 32, 32, 32)        128
_________________________________________________________________
```

```
conv2d_205 (Conv2D)          (None, 32, 32, 32)        9248
_________________________________________________________________
activation_255 (Activation)  (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_221 (Bat (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_101 (MaxPoolin (None, 16, 16, 32)        0
_________________________________________________________________
dropout_122 (Dropout)        (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_206 (Conv2D)          (None, 16, 16, 64)        18496
_________________________________________________________________
activation_256 (Activation)  (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_222 (Bat (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_207 (Conv2D)          (None, 16, 16, 64)        36928
_________________________________________________________________
activation_257 (Activation)  (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_223 (Bat (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_102 (MaxPoolin (None, 8, 8, 64)          0
_________________________________________________________________
dropout_123 (Dropout)        (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_208 (Conv2D)          (None, 8, 8, 128)         73856
_________________________________________________________________
activation_258 (Activation)  (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_224 (Bat (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_209 (Conv2D)          (None, 8, 8, 128)         147584
_________________________________________________________________
activation_259 (Activation)  (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_225 (Bat (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_103 (MaxPoolin (None, 4, 4, 128)         0
_________________________________________________________________
dropout_124 (Dropout)        (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_210 (Conv2D)          (None, 4, 4, 256)         295168
_________________________________________________________________
activation_260 (Activation)  (None, 4, 4, 256)         0
_________________________________________________________________
batch_normalization_226 (Bat (None, 4, 4, 256)         1024
_________________________________________________________________
```

```
conv2d_211 (Conv2D)          (None, 4, 4, 256)         590080
_________________________________________________________________
activation_261 (Activation)  (None, 4, 4, 256)         0
_________________________________________________________________
batch_normalization_227 (Bat (None, 4, 4, 256)         1024
_________________________________________________________________
max_pooling2d_104 (MaxPoolin (None, 2, 2, 256)         0
_________________________________________________________________
dropout_125 (Dropout)        (None, 2, 2, 256)         0
_________________________________________________________________
conv2d_212 (Conv2D)          (None, 2, 2, 512)         1180160
_________________________________________________________________
activation_262 (Activation)  (None, 2, 2, 512)         0
_________________________________________________________________
batch_normalization_228 (Bat (None, 2, 2, 512)         2048
_________________________________________________________________
conv2d_213 (Conv2D)          (None, 2, 2, 512)         2359808
_________________________________________________________________
activation_263 (Activation)  (None, 2, 2, 512)         0
_________________________________________________________________
batch_normalization_229 (Bat (None, 2, 2, 512)         2048
_________________________________________________________________
max_pooling2d_105 (MaxPoolin (None, 1, 1, 512)         0
_________________________________________________________________
dropout_126 (Dropout)        (None, 1, 1, 512)         0
_________________________________________________________________
flatten_25 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_50 (Dense)             (None, 128)               65664
_________________________________________________________________
activation_264 (Activation)  (None, 128)               0
_________________________________________________________________
batch_normalization_230 (Bat (None, 128)               512
_________________________________________________________________
dropout_127 (Dropout)        (None, 128)               0
_________________________________________________________________
dense_51 (Dense)             (None, 10)                1290
_________________________________________________________________
activation_265 (Activation)  (None, 10)                0
=================================================================
Total params: 4,787,626
Trainable params: 4,783,402
Non-trainable params: 4,224
_________________________________________________________________
None
```

By testing our model, we ended up having an overall accuracy, recall, and f1-score of 0.84. Generally, most of the error came from not being able to properly distinguish cats and dogs as well as smaller errors coming from having difficulty distinguishing animals in general. The model did relatively better in accurately classifying inanimate objects (ships, trucks, airplanes, and automobiles). The accuracy of predictions could be potentially improved by training the model without these inanimate objects and including more training data of the similar animals (cats and dogs). This might allow the model to find smaller differences between animals that may be of similar sizes or have similar features (like cats and dogs).