In [1]:

```
import os
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import time

%matplotlib inline
```

In [2]:

```
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
```

In [ ]:

```
#### Data ####

BATCH_SIZE = 64
DATASET = "CIFAR100"
NUM_CLASSES = 100

def normalize_data(X_train, X_test):
    _mean = X_train.mean(axis=(0, 1, 2), keepdims=True)
    _std = X_train.std(axis=(0, 1, 2), keepdims=True)
    X_train = (X_train - _mean) / _std
    X_test = (X_test - _mean) / _std
    return X_train, X_test

def get_tf_dataset_iter(data):
    dataset = tf.data.Dataset.from_tensor_slices(data)
    dataset = dataset.repeat(None)    # Repeat Indefinitely
    dataset = dataset.batch(BATCH_SIZE)
    dataset = dataset.prefetch(1)
    dataiter = iter(dataset)
    return dataiter
```

In [3]:

```
def get_data(dataset):
    if dataset == "CIFAR100":
        data = tf.keras.datasets.cifar100.load_data()
    elif dataset == "CIFAR10":
        data = tf.keras.datasets.cifar10.load_data()
    else:
        print ("Invalid Dataset!!")
        exit(-1)

    (X_train, y_train), (X_test, y_test) = data
    X_train = np.asarray(X_train, dtype=np.float32)
    y_train = np.asarray(y_train, dtype=np.int32).flatten()
    X_test = np.asarray(X_test, dtype=np.float32)
    y_test = np.asarray(y_test, dtype=np.int32).flatten()

    X_train, X_test = normalize_data(X_train, X_test)

    X_train, X_val = np.split(X_train, [int(0.98*len(X_train))])
    y_train, y_val = np.split(y_train, [int(0.98*len(y_train))])

    return X_train, y_train, X_val, y_val, X_test, y_test


X_train, y_train, X_val, y_val, X_test, y_test = get_data(DATASET)
X_train_iter = get_tf_dataset_iter(X_train)
y_train_iter = get_tf_dataset_iter(y_train)
X_val_iter = get_tf_dataset_iter(X_val)
y_val_iter = get_tf_dataset_iter(y_val)
X_test_iter = get_tf_dataset_iter(X_test)
y_test_iter = get_tf_dataset_iter(y_test)
```

In [4]:

```
def timeme(method):
    def wrapper(*args, **kw):
        startTime = int(round(time.time() * 1000))
        result = method(*args, **kw)
        endTime = int(round(time.time() * 1000))

        print("Execution Time:", endTime - startTime,'ms')
        return result

    return wrapper
```

In [5]:

```python
def get_accuracy(model, Xset, yset, Xiter, yiter):
    total_batches = (Xset.shape[0] // BATCH_SIZE) + 1
    num_correct, num_samples = 0, 0
    for batch_num in range(total_batches):
        batch_X = next(Xiter)
        batch_y = next(yiter).numpy()
        scores = model(batch_X, is_training = False)
        scores = scores.numpy()
        pred_y = scores.argmax(axis=1)
        num_samples += batch_X.shape[0]
        num_correct += (pred_y == batch_y).sum()
    accuracy = float(num_correct) / num_samples
    return 100*accuracy
```

In [31]:

```
EPOCH_ITERATIONS = 700
LEARNING_RATE = 3e-4
EXPERIMENT_ROOT = './experiment'
PRINT_FREQUENCY = 100

@timeme
# def train(CNN_class, num_epochs = 10, resume = False, plot_losses = True):
def train(model, num_epochs = 10, resume = False, plot_losses = True):
#     model = CNN_class(NUM_CLASSES)
    decay_start_iteration = 0.6 * (num_epochs * EPOCH_ITERATIONS)
    lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(LEARNING_RATE,
    optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)

    writer = tf.summary.create_file_writer(EXPERIMENT_ROOT)
    ckpt = tf.train.Checkpoint(step=tf.Variable(0), optimizer=optimizer, net=mo
    manager = tf.train.CheckpointManager(ckpt, EXPERIMENT_ROOT, max_to_keep=10)

    if resume:
        ckpt.restore(manager.latest_checkpoint)

    stop_training = False
    for epc in range(1, num_epochs + 1):
        print ("[Epoch {}]".format(epc))
        loss_list = []
        validation_accuracies = []
        for iter_num in range(EPOCH_ITERATIONS):
            batch_X = next(X_train_iter)
            batch_y = next(y_train_iter)
            with tf.GradientTape() as tape:
                scores = model(batch_X, is_training = True)
                losses = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=
                meanloss = tf.reduce_mean(losses)
                loss_list.append(meanloss)
                lossnp = losses.numpy()

            if iter_num % PRINT_FREQUENCY == 0:

                with writer.as_default():
                    tf.summary.scalar("loss", meanloss, step=iter_num)
                    tf.summary.histogram('losses',losses,step=iter_num)

                print('iter:{:6d}, loss min|avg|max: {:.3f}|{:.3f}|{:6.3f}, '
                      .format(iter_num,
                              float(np.min(lossnp)),
                              float(np.mean(lossnp)),
                              float(np.max(lossnp))), end="")
                validation_acc = get_accuracy(model, X_val, y_val, X_val_iter,
                validation_accuracies.append(validation_acc)
                print ('val acc: {:.2f} %'.format(validation_acc))

                if epc >= 3 and validation_acc <= 20.0:
                    print ("Circuit Breaker!! Validation accuracy too low")
                    stop_training = True
                    break

            grad = tape.gradient(meanloss, model.trainable_variables)
            optimizer.apply_gradients(zip(grad, model.trainable_variables))
            ckpt.step.assign_add(1)

```

```
        manager.save()

        if stop_training:
            break

        if plot_losses:
            plt.plot(loss_list, 'r')
            plt.grid(True)
            plt.title('Epoch {} Loss'.format(epc))
            plt.xlabel('MiniBatch number')
            plt.ylabel('MiniBatch loss')
            plt.show()

            plt.plot(validation_accuracies)
            plt.grid(True)
            plt.title('Epoch {} Validation Accuracies'.format(epc))
            plt.xlabel('MiniBatch number (100s)')
            plt.ylabel('Validation Accuracy')
            plt.show()
#     return model
```

CNN1

In [7]:

```
class CNN1(tf.keras.Model):
    def __init__(self, num_classes, input_shape = (32, 32, 3)):
        super().__init__()
        self.conv1 = tf.keras.layers.Conv2D(input_shape = input_shape, filters
                            padding = 'SAME', activation = tf.nn.relu, name = '
        self.conv2 = tf.keras.layers.Conv2D(filters = 32, kernel_size = [5,5],
                            padding = 'SAME', activation = tf.nn.relu, name = '
        self.flatten1 = tf.keras.layers.Flatten()
        self.fc1 = tf.keras.layers.Dense(512, name = 'fc1')
        self.fc2 = tf.keras.layers.Dense(num_classes, name = 'fc2')

    def call(self, batch_input, is_training):
        x = self.conv1(batch_input)
        x = self.conv2(x)
        x = self.flatten1(x)
        x = self.fc1(x)
        x = self.fc2(x)
        return x
```

CNN2

In [8]:

```
class CNN2(tf.keras.Model):
    def __init__(self, num_classes, input_shape = (32, 32, 3)):
        super().__init__()
        self.conv1 = tf.keras.layers.Conv2D(input_shape = input_shape, filters
                            padding = 'SAME', activation = tf.nn.relu, name = '
        self.conv2 = tf.keras.layers.Conv2D(filters = 32, kernel_size = [3,3],
                            padding = 'SAME', activation = tf.nn.relu, name = '
        self.flatten1 = tf.keras.layers.Flatten()
        self.fc1 = tf.keras.layers.Dense(512, name = 'fc1')
        self.fc2 = tf.keras.layers.Dense(num_classes, name = 'fc2')

    def call(self, batch_input, is_training):
        x = self.conv1(batch_input)
        x = self.conv2(x)
        x = self.flatten1(x)
        x = self.fc1(x)
        x = self.fc2(x)
        return x
```

CNN3

In [9]:

```
class CNN3(tf.keras.Model):
    def __init__(self, num_classes, input_shape = (32, 32, 3)):
        super().__init__()
        self.conv1 = tf.keras.layers.Conv2D(input_shape = input_shape, filters
                            padding = 'SAME', activation = tf.nn.relu, name = '
        self.conv2 = tf.keras.layers.Conv2D(filters = 64, kernel_size = [5,5],
                            padding = 'SAME', activation = tf.nn.relu, name = '
        self.flatten1 = tf.keras.layers.Flatten()
        self.fc1 = tf.keras.layers.Dense(512, name = 'fc1')
        self.fc2 = tf.keras.layers.Dense(num_classes, name = 'fc2')

    def call(self, batch_input, is_training):
        x = self.conv1(batch_input)
        x = self.conv2(x)
        x = self.flatten1(x)
        x = self.fc1(x)
        x = self.fc2(x)
        return x
```

CNN4

In [10]:

```
class CNN4(tf.keras.Model):
    def __init__(self, num_classes, input_shape = (32, 32, 3)):
        super().__init__()
        self.conv1 = tf.keras.layers.Conv2D(input_shape = input_shape, filters
                            padding = 'SAME', activation = tf.nn.relu, name = '
        self.max_pool1 = tf.keras.layers.MaxPooling2D(pool_size = [2,2], stride
        self.conv2 = tf.keras.layers.Conv2D(filters = 64, kernel_size = [3,3],
                            padding = 'SAME', activation = tf.nn.relu, name = '
        self.max_pool2 = tf.keras.layers.MaxPooling2D(pool_size = [2,2], stride
        self.flatten1 = tf.keras.layers.Flatten()
        self.fc1 = tf.keras.layers.Dense(512, name = 'fc1')
        self.bnorm1 = tf.keras.layers.BatchNormalization(axis=1,momentum=0.9,ep
        self.fc2 = tf.keras.layers.Dense(num_classes, name = 'fc2')

    def call(self, batch_input, is_training):
        x = self.conv1(batch_input)
        x = self.max_pool1(x)
        x = self.conv2(x)
        x = self.max_pool2(x)
        x = self.flatten1(x)
        x = self.fc1(x)
        x = self.bnorm1(x)
        x = self.fc2(x)
        return x
```

CNN5

In [11]:

```
class CNN5(tf.keras.Model):
    def __init__(self, num_classes, input_shape = (32, 32, 3)):
        super().__init__()
        self.conv1 = tf.keras.layers.Conv2D(input_shape = input_shape, filters
                            padding = 'SAME', activation = tf.nn.relu, name = '
        self.max_pool1 = tf.keras.layers.MaxPooling2D(pool_size = [2,2], stride
        self.conv2 = tf.keras.layers.Conv2D(filters = 64, kernel_size = [3,3],
                            padding = 'SAME', activation = tf.nn.relu, name = '
        self.max_pool2 = tf.keras.layers.MaxPooling2D(pool_size = [2,2], stride
        self.flatten1 = tf.keras.layers.Flatten()
        self.fc1 = tf.keras.layers.Dense(512, name = 'fc1')
        self.bnorm1 = tf.keras.layers.BatchNormalization(axis=1,momentum=0.9,ep
        self.fc2 = tf.keras.layers.Dense(num_classes, name = 'fc2')

    def call(self, batch_input, is_training):
        x = self.conv1(batch_input)
        x = self.max_pool1(x)
        x = self.conv2(x)
        x = self.max_pool2(x)
        x = self.flatten1(x)
        x = self.fc1(x)
        x = self.bnorm1(x, is_training)
        x = self.fc2(x)
        return x
```

CNN6

In [27]:

```
class CNN6(tf.keras.Model):
    def __init__(self, num_classes, input_shape = (32, 32, 3)):
        super().__init__()
        self.conv1 = tf.keras.layers.Conv2D(input_shape = input_shape, filters
                            padding = 'SAME', activation = tf.nn.relu, name = '
        self.bnorm1 = tf.keras.layers.BatchNormalization(axis = 1, momentum = 0

        self.conv2 = tf.keras.layers.Conv2D(filters = 64, kernel_size = [3,3],
                            padding = 'SAME', activation = tf.nn.relu, name = '
        self.bnorm2 = tf.keras.layers.BatchNormalization(axis = 1, momentum = 0

        self.max_pool1 = tf.keras.layers.MaxPooling2D(pool_size = [2,2], stride

        self.flatten1 = tf.keras.layers.Flatten()
        self.fc1 = tf.keras.layers.Dense(512, name = 'fc1')
        self.bnorm3 = tf.keras.layers.BatchNormalization(axis = 1, momentum = 0
        self.fc2 = tf.keras.layers.Dense(num_classes, name = 'fc2')

    def call(self, batch_input, is_training):
        x = self.conv1(batch_input)
        x = self.bnorm1(x, is_training)
        x = self.conv2(x)
        x = self.bnorm2(x, is_training)
        x = self.max_pool1(x)
        x = self.flatten1(x)
        x = self.fc1(x)
        x = self.bnorm3(x, is_training)
        x = self.fc2(x)
        return x
```

CNN7

In [13]:

```
class CNN7(tf.keras.Model):
    def __init__(self, num_classes, input_shape = (32, 32, 3)):
        super().__init__()
        self.conv1 = tf.keras.layers.Conv2D(input_shape = input_shape, filters
                            padding = 'SAME', activation = tf.nn.relu, name = '
        self.max_pool1 = tf.keras.layers.MaxPooling2D(pool_size = [2,2], stride
        self.conv2 = tf.keras.layers.Conv2D(filters = 128, kernel_size = [3,3],
                            padding = 'SAME', activation = tf.nn.relu, name = '
        self.max_pool2 = tf.keras.layers.MaxPooling2D(pool_size = [2,2], stride
        self.conv3 = tf.keras.layers.Conv2D(filters = 256, kernel_size = [3,3],
                            padding = 'SAME', activation = tf.nn.relu, name = '
        self.gpool1 = tf.keras.layers.GlobalAveragePooling2D()
        self.fc1 = tf.keras.layers.Dense(512, name = 'fc1')
        self.bnorm1 = tf.keras.layers.BatchNormalization(axis = 1, momentum = 0
        self.fc2 = tf.keras.layers.Dense(num_classes, name = 'fc2')

    def call(self, batch_input, is_training):
        x = self.conv1(batch_input)
        x = self.max_pool1(x)
        x = self.conv2(x)
        x = self.max_pool2(x)
        x = self.conv3(x)
        x = self.gpool1(x)
        x = self.fc1(x)
        x = self.bnorm1(x, is_training)
        x = self.fc2(x)
        return x
```

Experiments

Experiment #1 - CNN1

In [46]:

```
cnn1_model = CNN1(NUM_CLASSES)
train(cnn1_model, num_epochs = 30)
```

```
[Epoch 1]
iter:     0, loss min|avg|max: 4.066|4.752| 5.680, val acc: 1.07 %
iter:   100, loss min|avg|max: 1.429|4.040| 7.318, val acc: 11.23 %
iter:   200, loss min|avg|max: 0.281|3.568| 7.741, val acc: 17.29 %
iter:   300, loss min|avg|max: 0.425|3.223|10.827, val acc: 19.43 %
iter:   400, loss min|avg|max: 0.135|2.976| 6.332, val acc: 24.51 %
iter:   500, loss min|avg|max: 0.030|2.999| 9.676, val acc: 24.80 %

---------------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
 last)
<ipython-input-46-458b4dc60723> in <module>
      1 cnn1_model = CNN1(NUM_CLASSES)
----> 2 train(cnn1_model, num_epochs = 30)

<ipython-input-4-30fa9fee35c5> in wrapper(*args, **kw)
      2     def wrapper(*args, **kw):
      3         startTime = int(round(time.time() * 1000))
----> 4         result = method(*args, **kw)
      5         endTime = int(round(time.time() * 1000))
      6

<ipython-input-31-4740c8fef31d> in train(model, num_epochs, resume, pl
ot_losses)
     54                     break
     55
---> 56             grad = tape.gradient(meanloss, model.trainable_var
iables)
     57             optimizer.apply_gradients(zip(grad, model.trainabl
e_variables))
     58             ckpt.step.assign_add(1)

~/.local/lib/python3.5/site-packages/tensorflow_core/python/eager/back
prop.py in gradient(self, target, sources, output_gradients, unconnect
ed_gradients)
   1027         output_gradients=output_gradients,
   1028         sources_raw=flat_sources_raw,
-> 1029         unconnected_gradients=unconnected_gradients)
   1030
   1031     if not self._persistent:

~/.local/lib/python3.5/site-packages/tensorflow_core/python/eager/impe
rative_grad.py in imperative_grad(tape, target, sources, output_gradie
nts, sources_raw, unconnected_gradients)
     75       output_gradients,
     76       sources_raw,
---> 77       compat.as_str(unconnected_gradients.value))

~/.local/lib/python3.5/site-packages/tensorflow_core/python/eager/back
prop.py in _gradient_function(op_name, attr_tuple, num_inputs, inputs,
outputs, out_grads, skip_input_indices)
    139     return [None] * num_inputs
    140
--> 141   return grad_fn(mock_op, *out_grads)
    142
```

```
    143

~/.local/lib/python3.5/site-packages/tensorflow_core/python/ops/math_g
rad.py in _MatMulGrad(op, grad)
   1629   if not t_a and not t_b:
   1630     grad_a = gen_math_ops.mat_mul(grad, b, transpose_b=True)
-> 1631     grad_b = gen_math_ops.mat_mul(a, grad, transpose_a=True)
   1632   elif not t_a and t_b:
   1633     grad_a = gen_math_ops.mat_mul(grad, b)

~/.local/lib/python3.5/site-packages/tensorflow_core/python/ops/gen_ma
th_ops.py in mat_mul(a, b, transpose_a, transpose_b, name)
   5604         _ctx._context_handle, tld.device_name, "MatMul", name,
   5605         tld.op_callbacks, a, b, "transpose_a", transpose_a, "t
ranspose_b",
-> 5606         transpose_b)
   5607       return _result
   5608     except _core._FallbackException:

KeyboardInterrupt:
```

In [47]:

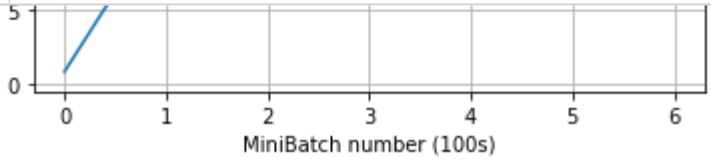
```
1 test_acc1 = get_accuracy(cnn1_model, X_test, y_test, X_test_iter, y_test_iter)
2 print ("CNN1 - Test Accuracy: {:.2f} %".format(test_acc2), end="\n\n")
3
4 cnn1_model.summary()
```

```
CNN1 - Test Accuracy: 30.16 %

Model: "cn_n1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1 (Conv2D)               multiple                  1216
_________________________________________________________________
conv2 (Conv2D)               multiple                  12832
_________________________________________________________________
flatten_13 (Flatten)         multiple                  0
_________________________________________________________________
fc1 (Dense)                  multiple                  16777728
_________________________________________________________________
fc2 (Dense)                  multiple                  51300
=================================================================
Total params: 16,843,076
Trainable params: 16,843,076
Non-trainable params: 0
_________________________________________________________________
```

Experiment #2 - CNN2

In [44]:

```
cnn2_model = CNN2(NUM_CLASSES)
train(cnn2_model, num_epochs = 30)
```




```
[Epoch 2]
iter:     0, loss min|avg|max: 0.007|2.647| 6.875, val acc: 29.79 %
iter:   100, loss min|avg|max: 0.043|2.798| 7.152, val acc: 29.88 %
iter:   200, loss min|avg|max: 0.004|2.500|12.414, val acc: 31.74 %
iter:   300, loss min|avg|max: 0.000|2.217| 6.415, val acc: 30.76 %
iter:   400, loss min|avg|max: 0.004|1.897| 7.133, val acc: 29.59 %
```

In [45]:

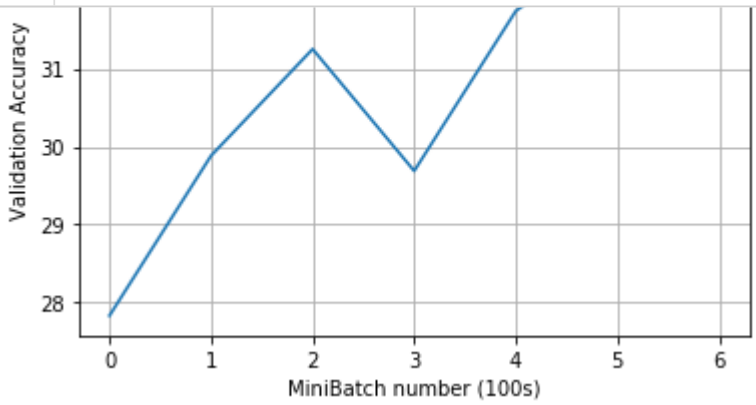
```
test_acc2 = get_accuracy(cnn2_model, X_test, y_test, X_test_iter, y_test_iter)
print ("CNN2 - Test Accuracy: {:.2f} %".format(test_acc2), end="\n\n")

cnn2_model.summary()
```

```
CNN2 - Test Accuracy: 30.16 %

Model: "cn_n2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1 (Conv2D)               multiple                  448
_________________________________________________________________
conv2 (Conv2D)               multiple                  4640
_________________________________________________________________
flatten_12 (Flatten)         multiple                  0
_________________________________________________________________
fc1 (Dense)                  multiple                  16777728
_________________________________________________________________
fc2 (Dense)                  multiple                  51300
=================================================================
Total params: 16,834,116
Trainable params: 16,834,116
Non-trainable params: 0
_________________________________________________________________
```

Experiment #3 - CNN3

In [42]:

```
cnn3_model = CNN3(NUM_CLASSES)
train(cnn3_model, num_epochs = 30)
```




```
[Epoch 3]
iter:     0, loss min|avg|max: 0.007|1.983|10.436, val acc: 33.59 %
```

In [43]:

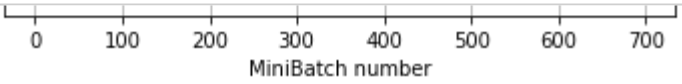
```
test_acc3 = get_accuracy(cnn3_model, X_test, y_test, X_test_iter, y_test_iter)
print ("CNN3 - Test Accuracy: {:.2f} %".format(test_acc3), end="\n\n")

cnn3_model.summary()
```

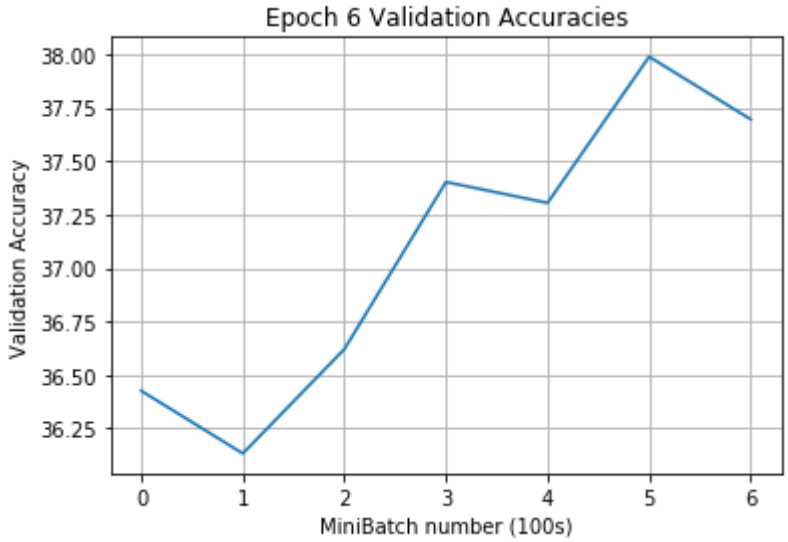
```
CNN3 - Test Accuracy: 32.90 %

Model: "cn_n3_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1 (Conv2D)               multiple                  2432
_________________________________________________________________
conv2 (Conv2D)               multiple                  51264
_________________________________________________________________
flatten_11 (Flatten)         multiple                  0
_________________________________________________________________
fc1 (Dense)                  multiple                  33554944
_________________________________________________________________
fc2 (Dense)                  multiple                  51300
=================================================================
Total params: 33,659,940
Trainable params: 33,659,940
Non-trainable params: 0
_________________________________________________________________
```

Experiment #4 - CNN4

In [40]:

```
cnn4_model = CNN4(NUM_CLASSES)
train(cnn4_model, num_epochs = 30)
```





In [41]:

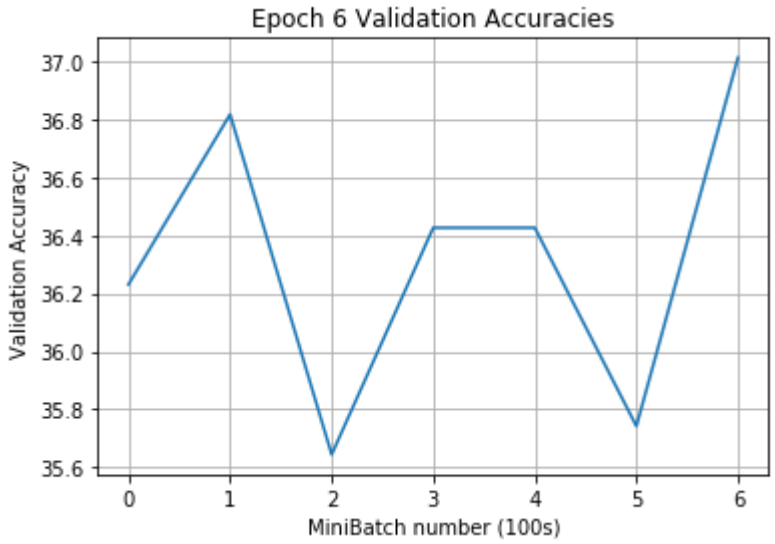
```
test_acc4 = get_accuracy(cnn4_model, X_test, y_test, X_test_iter, y_test_iter)
print ("CNN4 - Test Accuracy: {:.2f} %".format(test_acc4), end="\n\n")

cnn4_model.summary()
```

```
CNN4 - Test Accuracy: 37.42 %

Model: "cn_n4_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1 (Conv2D)               multiple                  896
_________________________________________________________________
max_pool1 (MaxPooling2D)     multiple                  0
_________________________________________________________________
conv2 (Conv2D)               multiple                  18496
_________________________________________________________________
max_pool2 (MaxPooling2D)     multiple                  0
_________________________________________________________________
flatten_10 (Flatten)         multiple                  0
_________________________________________________________________
fc1 (Dense)                  multiple                  2097664
_________________________________________________________________
bnorm1 (BatchNormalization)  multiple                  2048
_________________________________________________________________
fc2 (Dense)                  multiple                  51300
=================================================================
Total params: 2,170,404
Trainable params: 2,169,380
Non-trainable params: 1,024
_________________________________________________________________
```

Experiment #5 - CNN5

In [38]:

```
cnn5_model = CNN5(NUM_CLASSES)
train(cnn5_model, num_epochs = 30)
```




```
[Epoch 7]
iter:     0, loss min|avg|max: 0.001|1.543| 8.336, val acc: 36.82 %
iter:   100, loss min|avg|max: 0.005|1.142| 5.760, val acc: 35.94 %
```

In [39]:

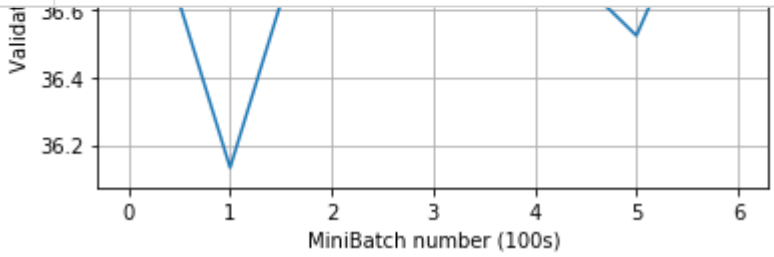
```
test_acc5 = get_accuracy(cnn5_model, X_test, y_test, X_test_iter, y_test_iter)
print ("CNN5 - Test Accuracy: {:.2f} %".format(test_acc5), end="\n\n")

cnn5_model.summary()
```

```
CNN5 - Test Accuracy: 39.20 %

Model: "cn_n5_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1 (Conv2D)               multiple                  896
_________________________________________________________________
max_pool1 (MaxPooling2D)     multiple                  0
_________________________________________________________________
conv2 (Conv2D)               multiple                  18496
_________________________________________________________________
max_pool2 (MaxPooling2D)     multiple                  0
_________________________________________________________________
flatten_9 (Flatten)          multiple                  0
_________________________________________________________________
fc1 (Dense)                  multiple                  2097664
_________________________________________________________________
bnorm1 (BatchNormalization)  multiple                  2048
_________________________________________________________________
fc2 (Dense)                  multiple                  51300
=================================================================
Total params: 2,170,404
Trainable params: 2,169,380
Non-trainable params: 1,024
_________________________________________________________________
```

Experiment #6 - CNN6

In [32]:

```
cnn6_model = CNN6(NUM_CLASSES)
train(cnn6_model, num_epochs = 30)
```




```
[Epoch 5]
iter:     0, loss min|avg|max: 0.001|0.770| 6.122, val acc: 36.04 %
```

In [34]:

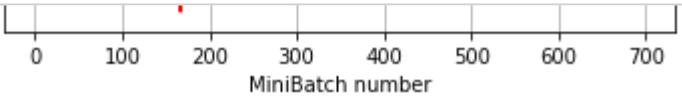
```
test_acc6 = get_accuracy(cnn6_model, X_test, y_test, X_test_iter, y_test_iter)
print ("CNN6 - Test Accuracy: {:.2f} %".format(test_acc6), end="\n\n")

cnn6_model.summary()
```

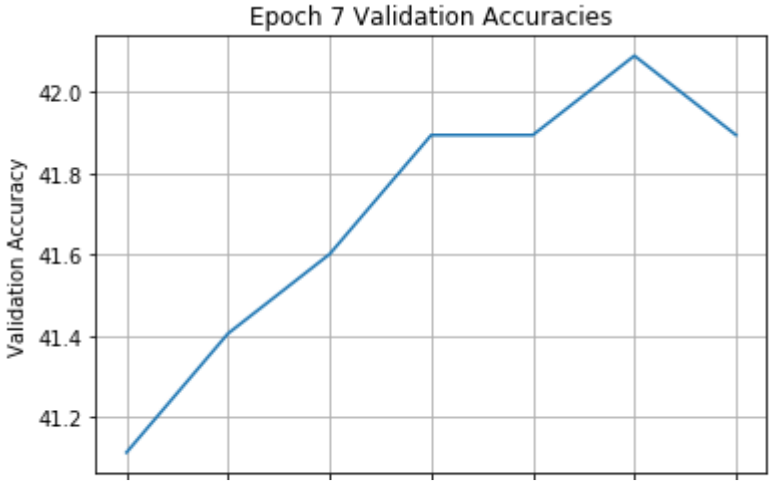
```
CNN6 - Test Accuracy: 35.65 %

Model: "cn_n6_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1 (Conv2D)               multiple                  896
_________________________________________________________________
bnorm1 (BatchNormalization)  multiple                  128
_________________________________________________________________
conv2 (Conv2D)               multiple                  18496
_________________________________________________________________
bnorm2 (BatchNormalization)  multiple                  128
_________________________________________________________________
max_pool1 (MaxPooling2D)     multiple                  0
_________________________________________________________________
flatten_8 (Flatten)          multiple                  0
_________________________________________________________________
fc1 (Dense)                  multiple                  8389120
_________________________________________________________________
bnorm3 (BatchNormalization)  multiple                  2048
_________________________________________________________________
fc2 (Dense)                  multiple                  51300
=================================================================
Total params: 8,462,116
Trainable params: 8,460,964
Non-trainable params: 1,152
_________________________________________________________________
```

Experiment #7 - CNN7

In [35]:

```
cnn7_model = CNN7(NUM_CLASSES)
train(cnn7_model, num_epochs = 30)
```





In [36]:

```
test_acc7 = get_accuracy(cnn7_model, X_test, y_test, X_test_iter, y_test_iter)
print ("CNN7 - Test Accuracy: {:.2f} %".format(test_acc7), end="\n\n")

cnn7_model.summary()
```

```
CNN7 - Test Accuracy: 41.67 %

Model: "cn_n7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1 (Conv2D)               multiple                  1792
_________________________________________________________________
max_pool1 (MaxPooling2D)     multiple                  0
_________________________________________________________________
conv2 (Conv2D)               multiple                  73856
_________________________________________________________________
max_pool2 (MaxPooling2D)     multiple                  0
_________________________________________________________________
conv3 (Conv2D)               multiple                  295168
_________________________________________________________________
global_average_pooling2d_1 ( multiple                  0
_________________________________________________________________
fc1 (Dense)                  multiple                  131584
_________________________________________________________________
bnorm1 (BatchNormalization)  multiple                  2048
_________________________________________________________________
fc2 (Dense)                  multiple                  51300
=================================================================
Total params: 555,748
Trainable params: 554,724
Non-trainable params: 1,024
_________________________________________________________________
```

Max. Test Accuracy = 41.67 %

CNN architecture - **CNN7**
Epochs: **7**
Iterations per epoch: **700**
Learning Rate: **3e-4**