Using flags in official models

1. **All common flags must be incorporated in the models.**

   Common flags (i.e. batch_size, model_dir, etc.) are provided by various flag definition functions, and channeled through `official.utils.flags.core`. For instance to define common supervised learning parameters one could use the following code:

   ```
   from absl import app as absl_app
   from absl import flags

   from official.utils.flags import core as flags_core


   def define_flags():
     flags_core.define_base()
     flags.adopt_key_flags(flags_core)


   def main(_):
     flags_obj = flags.FLAGS
     print(flags_obj)


   if __name__ == "__main__"
     absl_app.run(main)
   ```

2. **Validate flag values.**

   See the Validators section for implementation details.

   Validators in the official model repo should not access the file system, such as verifying that files exist, due to the strict ordering requirements.

3. **Flag values should not be mutated.**

   Instead of mutating flag values, use getter functions to return the desired values. An example getter function is `get_tf_dtype` function below:

   ```
   # Map string to TensorFlow dtype
   DTYPE_MAP = {
       "fp16": tf.float16,
       "fp32": tf.float32,
   }

   def get_tf_dtype(flags_obj):
     if getattr(flags_obj, "fp16_implementation", None) == "graph_rewrite":
       # If the graph_rewrite is used, we build the graph with fp32, and let the
       # graph rewrite change ops to fp16.
       return tf.float32
     return DTYPE_MAP[flags_obj.dtype]
   ```

```
def main(_):
  flags_obj = flags.FLAGS()

  # Do not mutate flags_obj
  # if flags_obj.fp16_implementation == "graph_rewrite":
  #   flags_obj.dtype = "float32" # Don't do this

  print(get_tf_dtype(flags_obj))
  ...
```